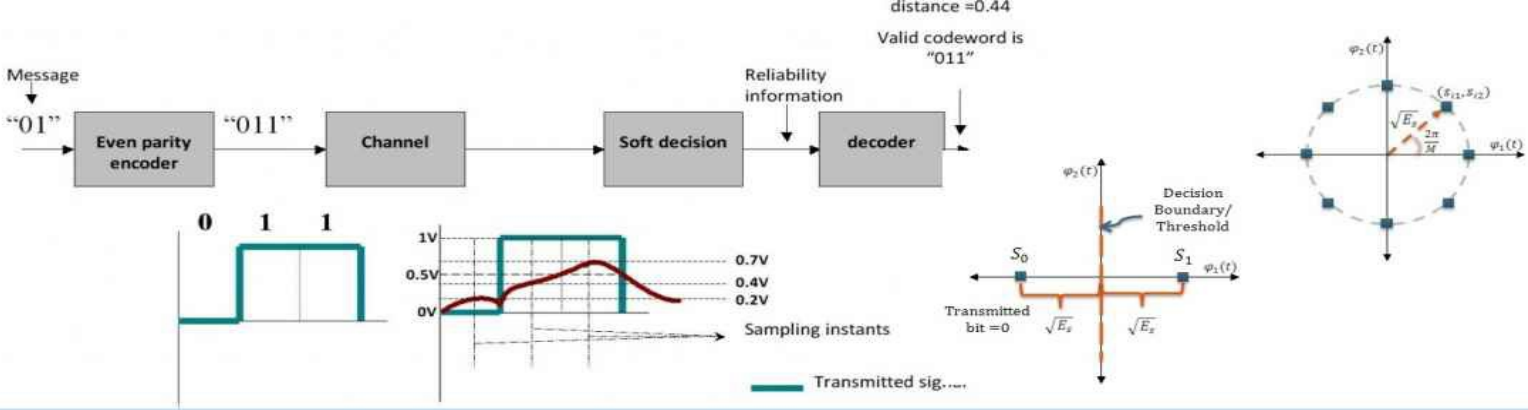
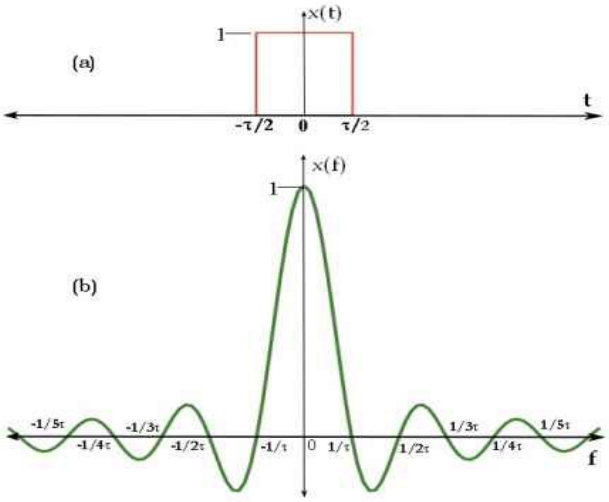
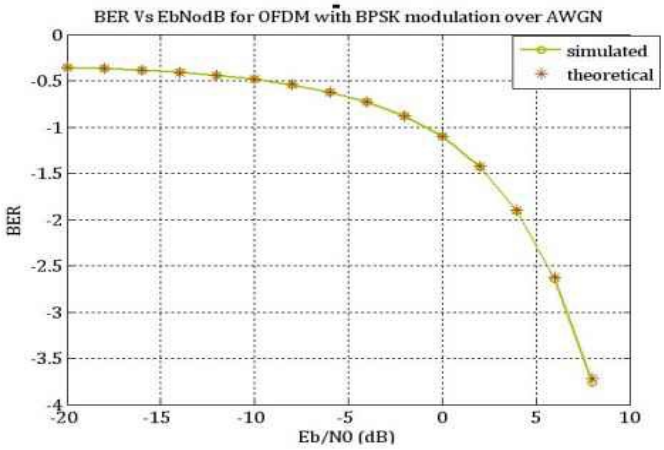
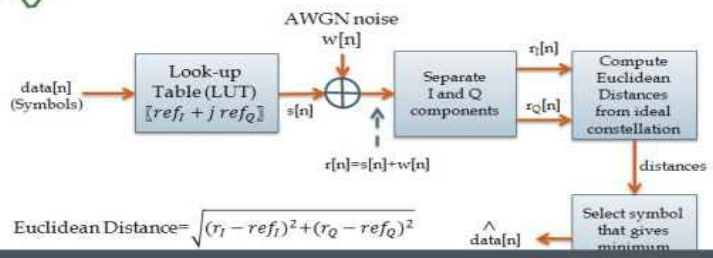


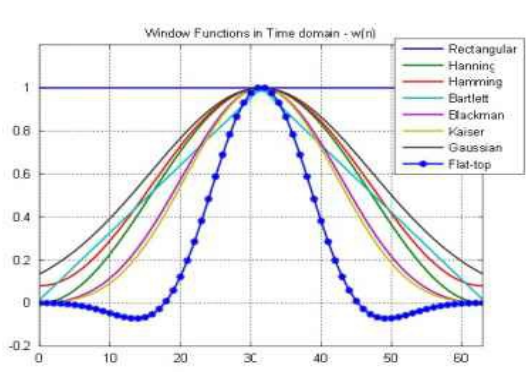
Simulation of Digital Communication Systems Using Matlab

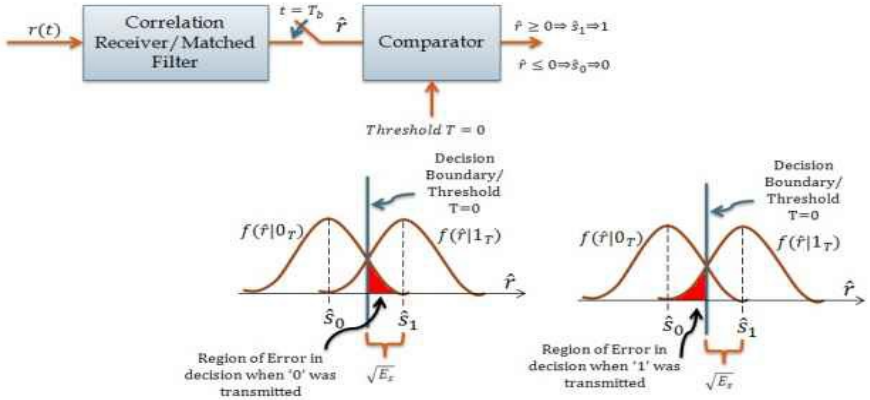






Mathuranathan Viswanathan





SIMULATION OF DIGITAL COMMUNICATION SYSTEMS USING MATLAB

By

Mathuranathan Viswanathan

SECOND EDITION FOR KINDLE

* * * * *

PUBLISHED BY:

Mathuranathan Viswanathan at Amazon

*****

This ebook is meant for students and instructors who are interested in simulation of signal processing and digital communication with Matlab. You should have a fair understanding of Matlab programming to begin with. Essential topics in digital communication are introduced to foster understanding of simulation methodologies. References are given in square brackets with in the text. Please refer the last section on references to get more details. The following manuscript is a result of five years of author's work and you are welcome to give feedback to make it better. Please check author's page (given at the end of this book) for contact info.

Acknowledgement: Thanks to Varsha Mathuranathan for editing and proof-reading this ebook.

*****

SIMULATION OF DIGITAL COMMUNICATION SYSTEMS USING MATLAB

*****

Table of Contents

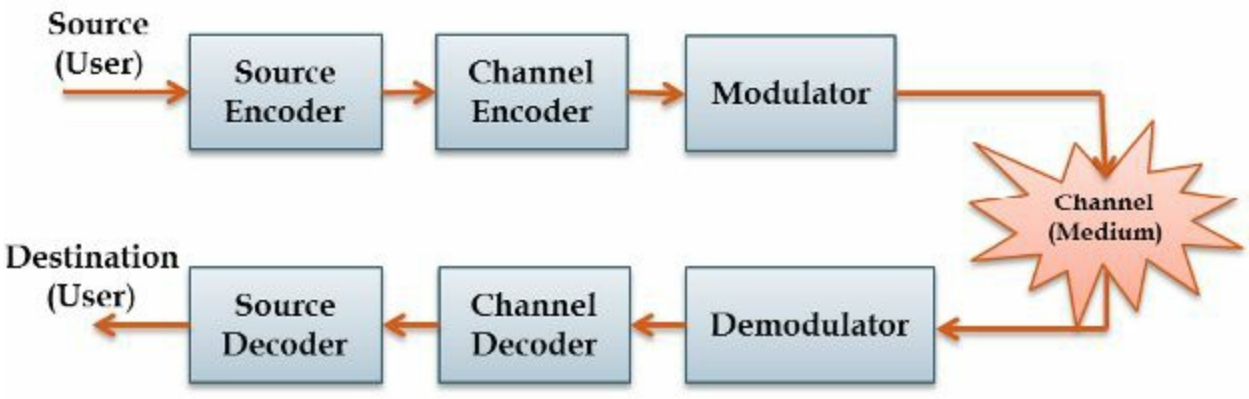
---End of Table of contents—

Chapter 1: Essentials of Digital Communication

1.1 Introduction to Digital Communication

Goals of Communication System Design:

Digital communication involves transmission of messages using finite alphabets (finite symbols) during finite time intervals (finite symbol interval). Any communication system (be it analog or digital in nature) in the electronic consumer market (be it hard disk drives, Compact Discs, telephony, mobile communication systems, etc..,) is made up of the following elements as represented in following figure.



The prime goals of a communication design engineer (one who designs a practical communication system) would be to

1) Reduce the bandwidth needed to send data.

Bandwidth, a limited and valuable resource, is the difference between the highest and the lowest frequency allocated for transmitting a message in any communication system. For example in GSM technology the typical bandwidth allocated for a single user is 200 KHz. More bandwidth provides space to transmit more data as well as more transmission rate (measured in bits per second - "bps"). The goal of reduced bandwidth is needed because of the growing bandwidth demands and the limited availability of communication spectrum. A downloading speed of 56Kbps was felt sufficient few years ago, but now it is not so. Hence it is essential to send more data in lesser bandwidth. This is achieved by compressing the data at the transmitting end and decompressing it at the receiving end. A "Source Encoder" and a "Source Decoder" serve this purpose.

2) To make data robust against harsh environments

Data will get corrupted when it is sent in harsh media (referred to as "channel"). For example mobile phones are operated in a very noisy environment in which the noise sources may be one or more of the following: interference from other mobile users, ignition noise, thermal noise, multipath interference and other man made noises. Channel coding is a technique to make the transmitted data robust to such noises, meaning that you can still recover your data (using a Channel Decoder) intact even if it is corrupted by certain amount of noise.

3) Send data over a long distance

Obviously data has to be sent over a long distance through any media used for/by the communication system. The media may be a simple twisted pair copper wires used in telephone networks or the air media in the case of a mobile or satellite communication system. In the physical world, it is not possible to send a signal (carrying data) over infinite distance. According to the inverse square law

of distance the intensity of the transmitted signal is inversely proportional to the square of the distance.

$$Signal\ Intensity \propto \frac{1}{distance^2}$$

The inverse square law of distance works at every nook and corner of the world to increasingly attenuate the signal's intensity over the distance and eventually kills the signal completely. Data can travel long distances if it has more energy. Now the challenge is to increase the energy of the signal so that it can travel the intended long distance.

A signal sent over a medium is essentially an electromagnetic wave. According to Planck–Einstein equation, the energy of a photon and the frequency of the associated electromagnetic wave are related by

$$E = h\nu$$

where E= Energy of the transmitted signal, h=Planck's constant and ν=frequency of transmission).

The above mentioned equation implies that the energy of the signal can be increased by increasing the frequency of transmission. Equivalently the frequency of the data has to be shifted from lower frequency region to higher frequency region. This is achieved by Modulation. Demodulation is the complementary operation that restores the original frequency contents of a message.

Source Coding and Decoding:

Source coding, the first block in the communication system architecture shown in the previous figure, is the process of encoding the information using lesser number of bits than the uncoded version of the information. Essentially it is the other name for compression. All data compression techniques can be classified under two categories namely lossless compression techniques and lossy compression techniques. In lossless compression the exact original data can be reconstructed from compressed data. But in lossy compression some errors exist after de-compression, but those errors are not obvious or perceivable. A Few lossless and lossy compression techniques are listed below.

Lossless compression Techniques:

1) LZW (Lempel Ziv Welch) coding – algorithm used in PDF documents [ZivMay1977], [ZivSep1977], [Welch1985].

2) Huffman coding [Huff1952] – used widely as the final coding stage

3) Shannon-Fano coding [Fano1949] – used in IMPLODE compression method used for ZIP file formats.

4) Run Length encoding [Golomb1966] – used in FAX machines.

5) Golomb Coding – used in image compression – (implemented in Rice Algorithm for image compression) [Rice1979]

Lossy Compression Techniques:

1) JPEG [William1993] – Image compression technique (an implementation of Discrete Cosine Transform (DCT))

2) MPEG [WebMPEG] – Motion picture compression technique

3) A-Law and Mu-Law compression [WebITUG711] – Used in Audio compression

4) Linear Predictive Coding (LPC) – Used in Speech signal processing [Deng2003]

5) RELP (Residually Excited LPC), CELP (Codebook Excited LPC) – variants of LPC used in GSM and CDMA for voice compression.

Channel Coding and Decoding:

The next block in a communication system is the channel coding block. There is an important difference between channel coding and source coding. Source coding attempts to compress the data to improve bandwidth utilization, whereas, channel coding attempts to add redundancy to the data to make it more reliable (which reduces data rate) and therefore more robust against the channel noise. Channel coding reduces the data rate and improves the reliability of the system.

Steps in Channel Coding Design:

1) Identify the Channel or the medium of communication

2) Model the channel according to its nature or choose from pre-defined models which best suits the actual environment.

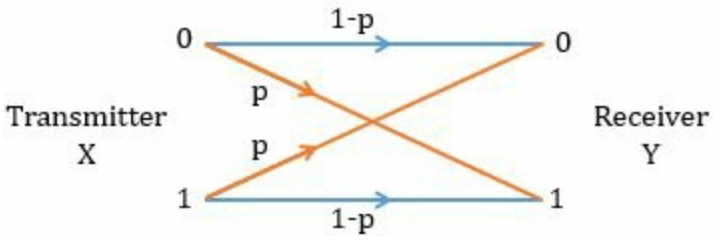
3) Decide over the type of coding strategy which will give best/required performance results by performing simulations

Some Channel Models:

Several models of channels were developed to design a communication system according to the possible type of channel one may use. Two of them are listed here.

Binary Symmetric Channel (BSC):

In this model, the transmitter sends a bit and the receiver receives it. Suppose if there exists a probability for this bit getting flipped, then it is referred to as a Binary Symmetric Channel. Under this model, the probability of erroneous reception is 'p' and the probability of correct reception is given by 1-p.This situation can be diagrammatically represented as shown in following figure



Given the transmitted bit represented by 'X' and received bit represented by Y, in terms of conditional probability, the Binary Symmetric Channel can be represented as

$$P(Y = 0|X = 0) = 1 - P$$
$$P(Y = 0|X = 1) = P$$
$$P(Y = 1|X = 0) = P$$
$$P(Y = 1|X = 1) = 1 - P$$

The above conditional probability specifies that the probability of erroneous reception (sent X=0 and received Y=1 or vice versa) is ‘p’ and the probability of correct reception is ‘1-p’.

Additive White Gaussian Noise Channel (AWGN):

In this model, the channel noise is assumed to have Gaussian nature and is additive. Compared to other equivalent channels, the AWGN channel does the maximum bit corruption and the systems designed to provide reliability in AWGN channel is assumed to give best performance results in other real-world channels. But the real performance may vary. The AWGN channel is a good model for many satellite and deep space communication links. In serial data communications, the AWGN mathematical model is used to model the timing error caused by random jitter. The distortion incurred by transmission over a lossy medium is modeled as the addition of a zero-mean Gaussian random value to each transmitted bit.

Channel Coding Design Approach:

The design approach that is widely used is called Forward Error Correction (FEC). This error correction technique is used to send data over unreliable noisy channels. The transmitted information is added with redundant bits using Error Correction Coding (ECC), otherwise called “channel coding”. This approach allows us to detect and correct the bit errors in the receiver without the need for retransmission. It is important to bear in mind that the correction and detection of errors are not absolute but rather statistical. Thus, one of our goals is to minimize the BER (Bit Error Rate) given a channel with certain noise characteristics and bandwidth.

In this method, K original bits, which are also called informational bits, are replaced with N>K new bits called “coded bits” or “code words”. The difference N-K represents the number of redundant bits added to the informational bits. Error Control Coding techniques are used to produce the code words from the information bits. The codewords carry with them an inherent potential (to certain extent) to recover from the distortions induced by the channel noise. The corresponding decoding technique in the receiver uses the redundant information in the codeword and tries to restore the original information, thereby providing immunity against the channel noise. There are two general schemes for channel coding: Linear Block Codes and (Linear) Convolution Codes. There exist even other sophisticated schemes/categories like Trellis Coded Modulation (TCM) which combines both the channel encoder and parts of the modulator into a single entity, non-linear channel coding techniques etc. To keep things simple, we will not go into the jungle of advanced channel coding techniques.

******

Back to Table of Contents

1.2 Sampling Theorem – Baseband Sampling

“Nyquist-Shannon Sampling Theorem” is the fundamental base over which all the digital processing techniques are built.

Processing a signal in digital domain gives several advantages (like immunity to temperature drift, accuracy, predictability, ease of design, ease of implementation etc..,) over analog domain

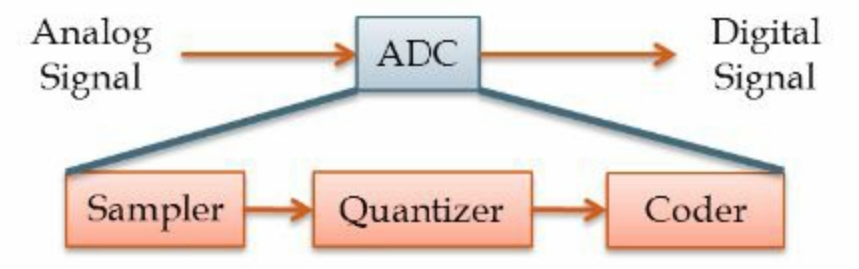
processing.

Analog to Digital conversion:

In analog domain, the signal that is of concern is continuous in both time and amplitude. The process of discretization of the analog signal in both time domain and amplitude levels yields the equivalent digital signal.

The conversion of analog to digital domain is a three step process

1) Discretization in time – Sampling
2) Discretization of amplitude levels – Quantization
3) Converting the discrete samples to digital samples – Coding/Encoding



The sampling operation samples ("chops") the incoming signal at regular intervals called "Sampling Rate" (denoted by T_S). Sampling Rate is determined by Sampling Frequency (denoted by F_S) as

$$T_s = \frac{1}{F_s}$$

Let's consider the following logical questions:

* Given a real world signal, how do we select the sampling rate in order to faithfully represent the signal in digital domain?

* Are there any criteria for selecting the sampling rate?

* Will there be any deviation if the signal is converted back to analog domain?

Answer: Consult the "Nyquist-Shannon Sampling Theorem" to select the sampling rate or sampling frequency.

Nyquist-Shannon Sampling Theorem:

The following sampling theorem is the exact reproduction of text from Shannon's classic paper [Shannon1949],

"If a function f(t) contains no frequencies higher than W cps, it is completely determined by giving its ordinates at a series of points spaced 1/2W seconds apart."

Sampling Theorem mainly falls into two categories:

1) Baseband Sampling – Applied for signals in the baseband (useful frequency components extending from 0Hz to some F_m Hz)

2) Band pass Sampling – Applied for signals whose frequency components extent from some F_1 Hz to F_2Hz (where $F_2 > F_1$)

In simple terms, the Nyquist Shannon Sampling Theorem for Baseband can be explained as follows

Baseband Sampling:

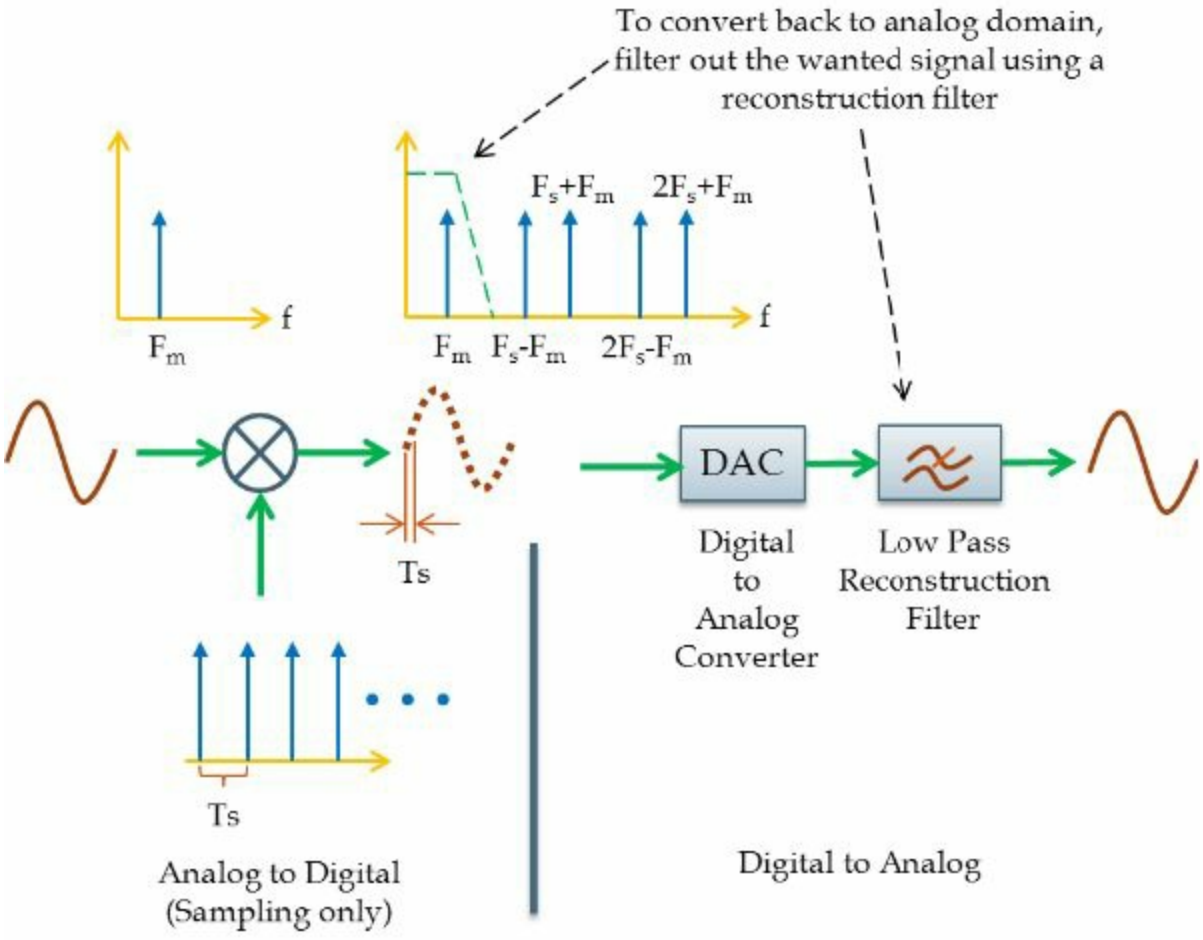
If the signal is confined to a maximum frequency of F_m Hz, in other words, the signal is a baseband signal (extending from 0 Hz to maximum F_m Hz).

For a faithful reproduction and reconstruction of an analog signal that is confined to a maximum frequency F_m, the signal should be sampled at a Sampling frequency (F_S) that is greater than or equal to twice the maximum frequency of the signal.

$$F_s \geq 2F_m$$

Consider a 10Hz sine wave in analog domain. The maximum frequency present in this signal is F_m=10 Hz (no doubt about it!!!). Now, to satisfy the sampling theorem that is stated above and to have a faithful representation of the signal in digital domain, the sampling frequency can be chosen as F_S >=20 Hz. That is, we are free to choose any number above 20 Hz. Higher the sampling frequency higher is the accuracy of representation of the signal. Higher sampling frequency also implies more samples, which implies more storage space or more memory requirements.

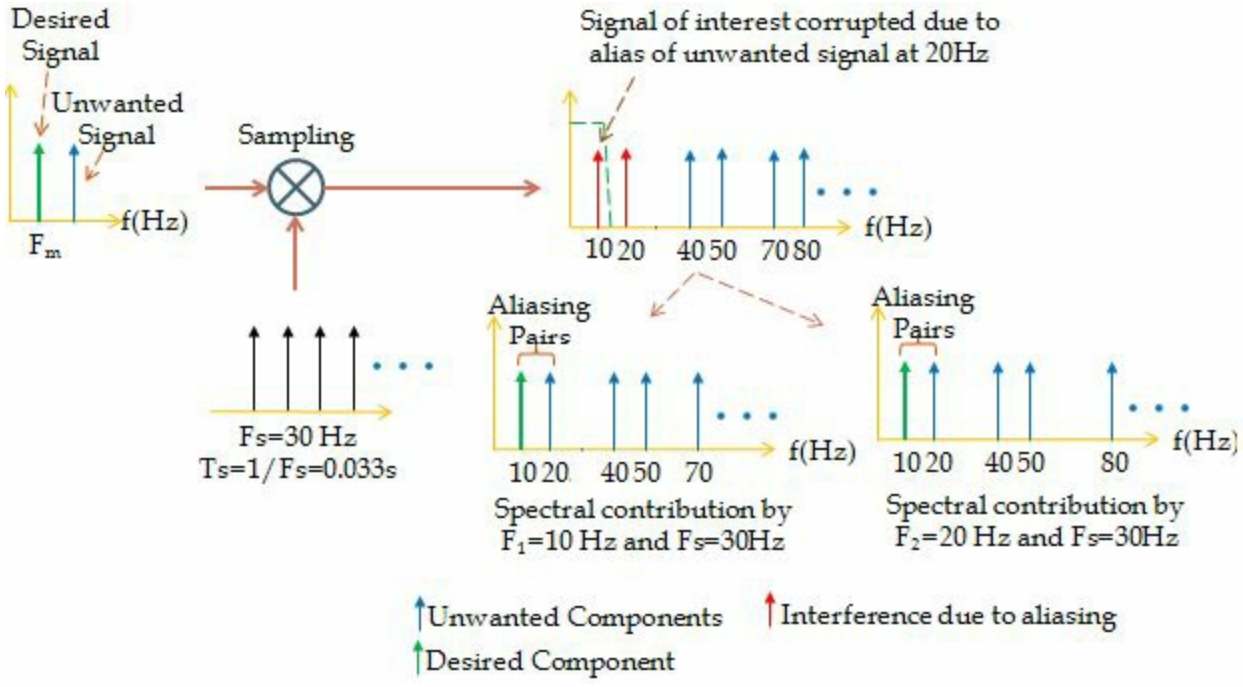
In time domain, the process of sampling can be viewed as multiplying the signal with a series of pulses ("pulse train) at regular intervals – T_S. In frequency domain, the output of the sampling process gives the following components – F_m (original frequency content of the signal), $F_S \pm F_m$, $2F_S \pm F_m$, $3F_S \pm F_m$, $4F_S \pm F_m$ and so on.



Now the sampled signal contains lots of unwanted frequency components ($F_S \pm F_m$, $2F_S \pm F_m$…). If we want to convert the sampled signal back to analog domain, all we need to do is to filter out those unwanted frequency components by using a "reconstruction" filter (In this case it is a low pass filter) that is designed to select only those frequency components that are up to F_m Hz. The above process mentions only the sampling part which samples the incoming analog signal at regular intervals. Actually a quantizer will follow the sampler which will discretize ("quantize") amplitude levels of the sampled signal. The quantized amplitude levels are sent to an encoder that converts the discrete amplitude levels to binary representation (binary data). So when converting the binary data back to analog domain, we need a Digital to Analog Converter (DAC) that converts the binary data to analog signal. Now the converted signal after the DAC contains the same unwanted frequencies as well as the wanted component. Thus a reconstruction filter with proper cut-off frequency has to be placed after the DAC to filter out only the wanted components.

Aliasing and Anti-aliasing:

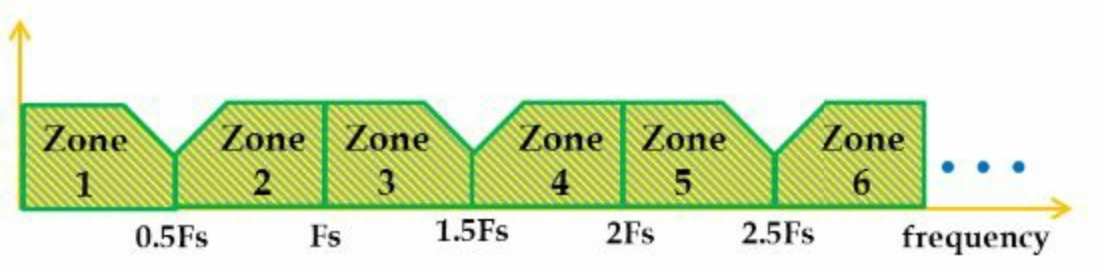
Consider a signal with two frequency components f_1=10 Hz – which is our desired signal and f_2=20 Hz – which is a noise. Let's say we sample the signal at 30 Hz. The first frequency component f_1=10 Hz will generate following frequency components at the output of the multiplier (sampler) – 10 Hz, 20 Hz, 40 Hz, 50 Hz, 70 Hz and so on. The second frequency component f_2=20 Hz will generate the following frequency components at the output of the multiplier – 20 Hz, 10 Hz, 50 Hz, 40 Hz, 80 Hz and so on.



Note the 10 Hz component that is generated by f_2=20 Hz. This 10 Hz component (which is a manifestation of noisy component f_2=20 Hz) will interfere with our original f1=10 Hz component and both are indistinguishable. This 10 Hz component is called an "alias" of the original component f_2=20 Hz (noise). Similarly the 20 Hz component generated by f_1=10 Hz component is an "alias" of f_1=10 Hz component. This 20 Hz alias of f_1=10 Hz will interfere with our original component f_2=20 Hz and are indistinguishable. We do not need to care about the interference that occurs at 20Hz since it is a noise and any way it has to be eliminated. But we need to do something about the aliasing component generated by the f_2=20 Hz. Since this is a noise component, the aliasing component generated by this

noise will interfere with our original f_1=10 Hz component and will corrupt it.

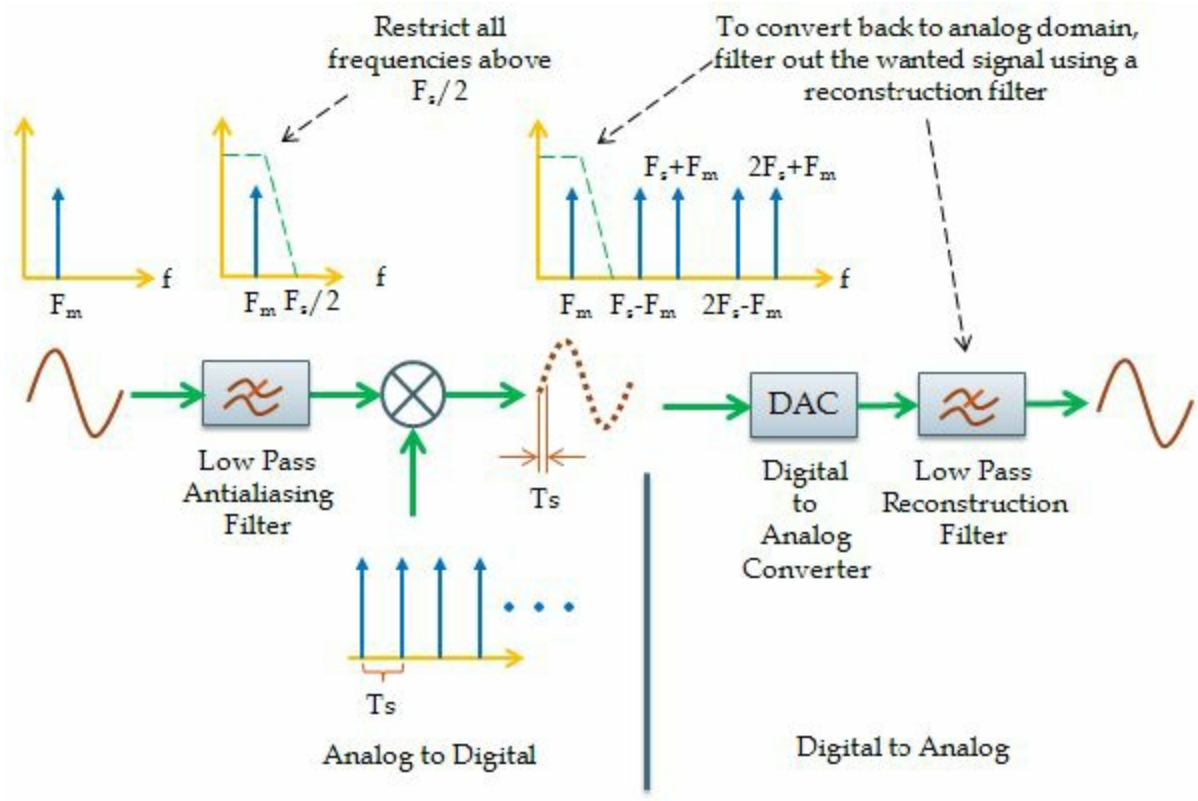
Aliasing depends on the sampling frequency and its relationship with the frequency components. If we sample a signal at F_S, all the frequency components from $F_S/2$ to Fs will be alias of frequency components from 0 to $F_S/2$ and vice versa. This frequency – $F_S/2$ is called "Folding frequency" since the frequency components from $F_S/2$ to F_S folds back itself and interferes with the components from 0Hz to $F_S/2$ Hz and vice versa. Actually the aliasing zones occur on the either sides of $0.5F_S$, $1.5F_S$, $2.5F_S$, $3.5F_S$ etc… All these frequencies are also called "Folding Frequencies" that causes frequency reversal. Similarly aliasing also occurs on either side of F_S, $2F_S$, $3F_S$, $4F_S$… without frequency reversals. The following figure illustrates the concept of aliasing zones.



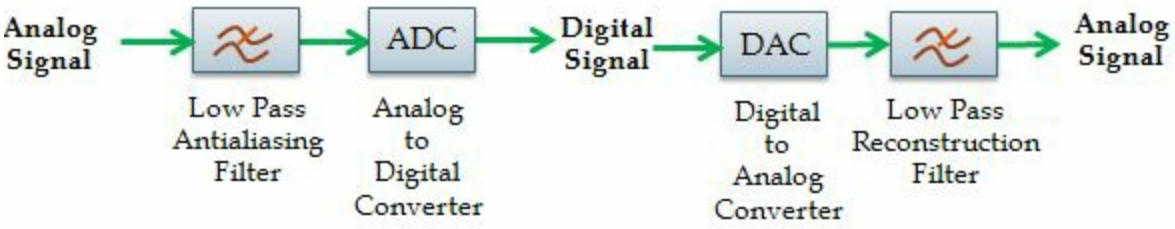
In the above figure, zone 2 is just a mirror image of zone 1 with frequency reversal. Similarly zone 2 will create aliases in zone 3 (without frequency reversal); zone 3 creates mirror image in zone 4 with frequency reversal and so on…

In the example above, the folding frequency was at $F_S/2$=15 Hz, so all the components from 15 Hz to 30 Hz will be the alias of the components from 0 Hz to 15 Hz. Once the aliasing components enter our band of interest, it is impossible to distinguish between original components and aliased components and as a result, the original content of the signal will be lost. In order to prevent aliasing, it is necessary to remove those frequencies that are above $F_S/2$ before sampling the signal. This is achieved by using an "anti-aliasing" filter that precedes the analog to digital converter.

An anti-aliasing filter is designed to restrict all the frequencies above the folding frequency $F_S/2$ and therefore avoids aliasing that may occur at the output of the multiplier otherwise.



Thus, a complete design of analog to digital conversion contains an anti-aliasing filter preceding the ADC and the complete design of digital to analog conversion contains a reconstruction filter succeeding the DAC.



Note: Remember that both the anti-aliasing and reconstruction filters are analog filters since they operate on analog signals. So it is imperative that the sampling rate has to be chosen carefully to relax the requirements for the anti-aliasing and reconstruction filters.

Effects of Sampling Rate:

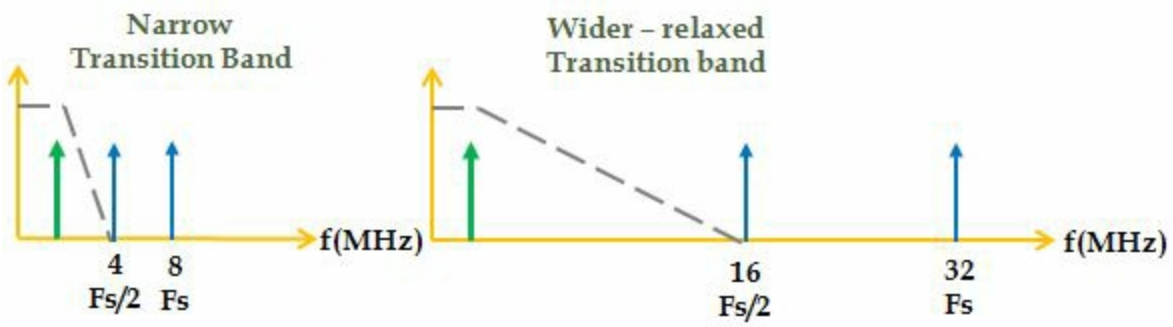
Consider a sinusoidal signal of frequency F_m=2 MHz. Let's say that we sample the signal at F_S=8MHz (F_S>=2*F_m). The factor F_S/F_m is called "over-sampling factor". In this case we are over-sampling the signal by a factor of F_m=8 MHz/2 MHz = 4. Now the folding frequency will be at $F_S/2$ = 4 MHz. Our anti-aliasing filter has to be designed to strictly cut off all the frequencies above 4 MHz to prevent aliasing.

In practice, ideal brick wall response for filters is not possible. Any filter will have a transition band between pass-band and stop-band. Sharper/faster roll off transition band (or narrow transition band) filters are always desired. But such filters are always of high orders. Since both the anti-aliasing and reconstruction filters are analog filters, high order filters that provide faster roll-off transition bands are expensive (Cost increases proportionally with filter order). The system also gets bulkier with increase in filter order. Therefore, to build a relatively cheaper system, the filter requirement (in-terms of width of the transition band) has to be relaxed. This can be done by increasing the sampling

rate or equivalently the over-sampling factor. When the sampling rate (F_S) is increased, the distance between the maximum frequency content F_m and $F_S/2$ will increase. This increase in the gap between the maximum frequency content of the signal and $F_S/2$ will ease requirements on the transition bands of the anti-aliasing analog filter.

The following figure illustrates this concept. If we use a sampling frequency of F_S=8 MHz (over-sampling factor = 4), the transition band is narrower and it calls for a higher order anti-aliasing filter If we increase the sampling frequency to F_S=32 MHz (over-sampling factor = 32 MHz/2 MHz=16), the distance between the desired component and $F_S/2$ has greatly increased that it facilitates a relatively inexpensive anti-aliasing filter with a wider transition band.

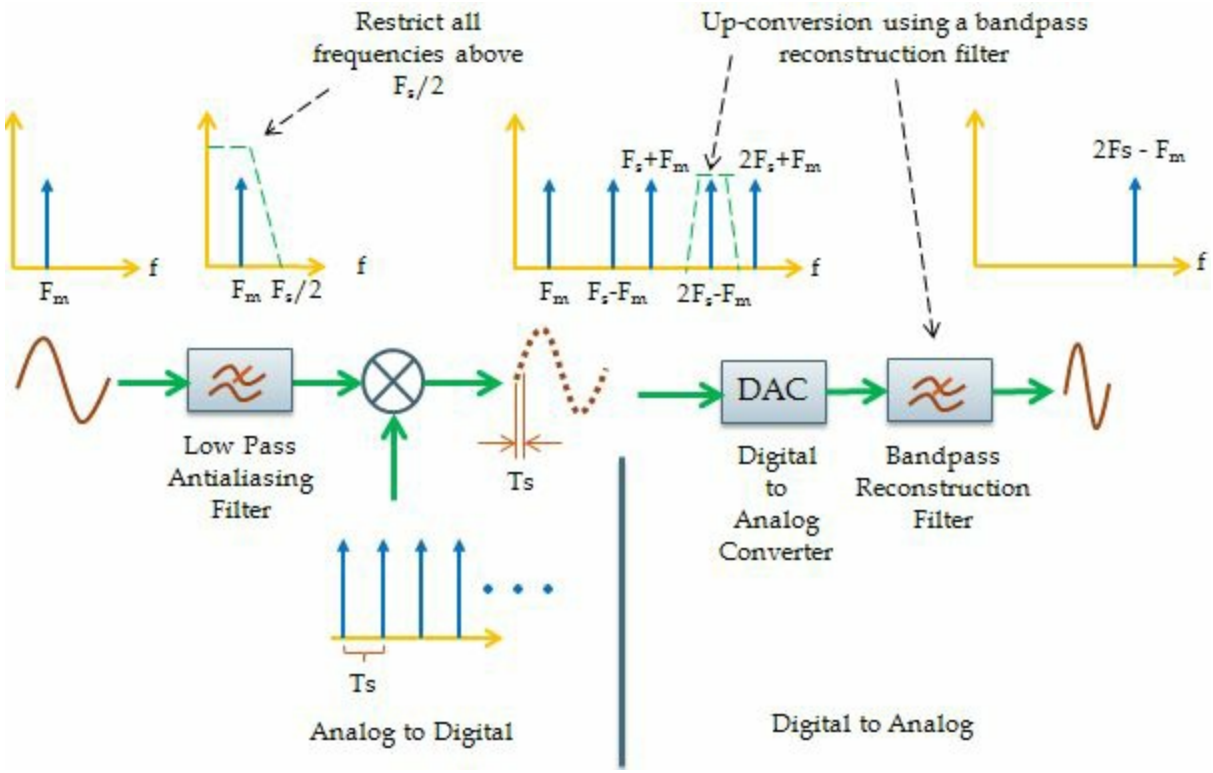
Thus, increasing the sampling rate of the ADC facilitates a simpler lower-order anti-aliasing filter and reconstruction filter. However, increase in the sampling rate calls for a faster sampler which makes ADC expensive. It is necessary to compromise and to strike balance between the sampling rate and the requirement of the anti-aliasing/reconstruction filter.



Application: Up-conversion

In the above examples, the reconstruction filter was conceived as a low pass filter that is designed to pass only the baseband frequency components after reconstruction. Remember that any frequency component present in zone 1 will be reflected in all the zones (with frequency reversals in even zones and without frequency reversals in odd zones). So, if we design the reconstruction filter to be a band pass filter, then selecting the reflected frequencies in any of the zones except zone 1, we achieve up-conversion.

In any communication system, the digital signal that comes out of a digital signal processor cannot be transmitted across as such. The processed signal (which is in the digital domain) has to be converted to analog signal and the analog signal has to be translated to an appropriate frequency of operation that fits the medium of transmission. For example, in an RF system, a baseband signal is converted to higher frequency (up-conversion) using a multiplier and oscillator. Then the high frequency signal is transmitted across the medium. If we have a band-pass reconstruction filter at the output of the DAC, we can directly achieve up-conversion (which saves us from using a multiplier and oscillator). The following figure illustrates this concept.



******

Back to Table of Contents

1.3 Sampling Theorem – Bandpass or Intermediate or Under Sampling

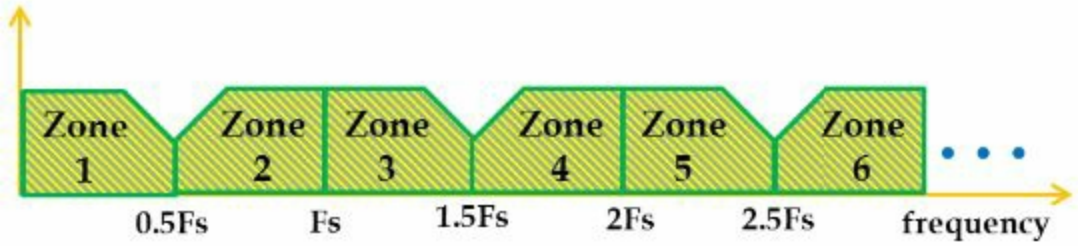
If the signal is a bandpass signal with bandwidth F_b (Bandwidth is simply the difference between the lowest and the highest frequency present in the signal).

"In order for a faithful reproduction and reconstruction of a bandpass analog signal with bandwidth – F_b, the signal should be sampled at a Sampling frequency (F_S) that is greater than or equal to twice the maximum bandwidth of the signal."

$$F_s \geq 2F_m$$

Consider a bandpass signal extending from 150 Hz to 200 Hz. The bandwidth of this signal is F_b=200 Hz-150 Hz=50 Hz. In order to faithfully represent the above signal in the digital domain the sampling frequency must be F_S>=100 Hz (2*F_b). Note that the sampling frequency 100 Hz is far below the maximum content of the signal (which is 200 Hz). That is why bandpass sampling is also called "under-sampling". As long as the sampling frequency is greater than or equal to twice the bandwidth of the signal, the reconstruction back to analog domain will be error free.

Going back to the aliasing zone figure, if the signal of interest is in the zone other than zone 1, it is called a bandpass signal and the sampling operation is called "Intermediate Sampling" or "Harmonic Sampling" or "Under Sampling" or "Bandpass Sampling".



Note that zone 1 is a mirror image of zone 2 (with frequency reversal). Similarly zone 3 is a mirror image of zone 4 etc. Also, any signal in zone 1 will be reflected in zone 2 with frequency reversal which in turn will be copied in zone 3 and so on.

Let's say the signal of interest lies in zone 2. This will be copied in all the other zones. Zone 1 also contains the sampled signal with frequency reversal which can be corrected by reversing the order of FFT in digital domain.

No matter in which zone the signal of interest lies, zone 1 always contains the signal after sampling operation is performed. If the signal of interest lies in any of the even zones, zone 1 contains the sampled signal with frequency reversal. If the signal of interest lies in any of the odd zones, zone 1 contains the sampled signal without frequency reversal.

Example:

Consider an AM signal centered at carrier frequency 1 MHz, with two components offset by 10 KHz – 0.99 MHz and 1.01 MHz. So the AM signal contains three frequency components at 0.99 MHz, 1 MHz and 1.01 MHz.

Our desire is to sample the AM signal. As with the usual sampling theorem (baseband), we know that if we sample the signal at twice the maximum frequency i.e F_S>=2*1.01 MHz=2.02 MHz there should be no problem in representing the analog signal in digital domain.

By the bandpass sampling theorem, we do not need to use a sampler running at F_S>=2.02 MHz. Faster sampling implies more cost. By applying the bandpass sampling theorem, we can use a slower sampler and reduce the cost of the system. The bandwidth of the signal is 1.01 MHz-0.99 MHz = 20 KHz. So, just sampling at F_S>= 2 * 20 KHz = 40 KHz will convert the signal to digital domain properly (if F_S >= 2.02 MHz is used according to baseband sampling theorem).

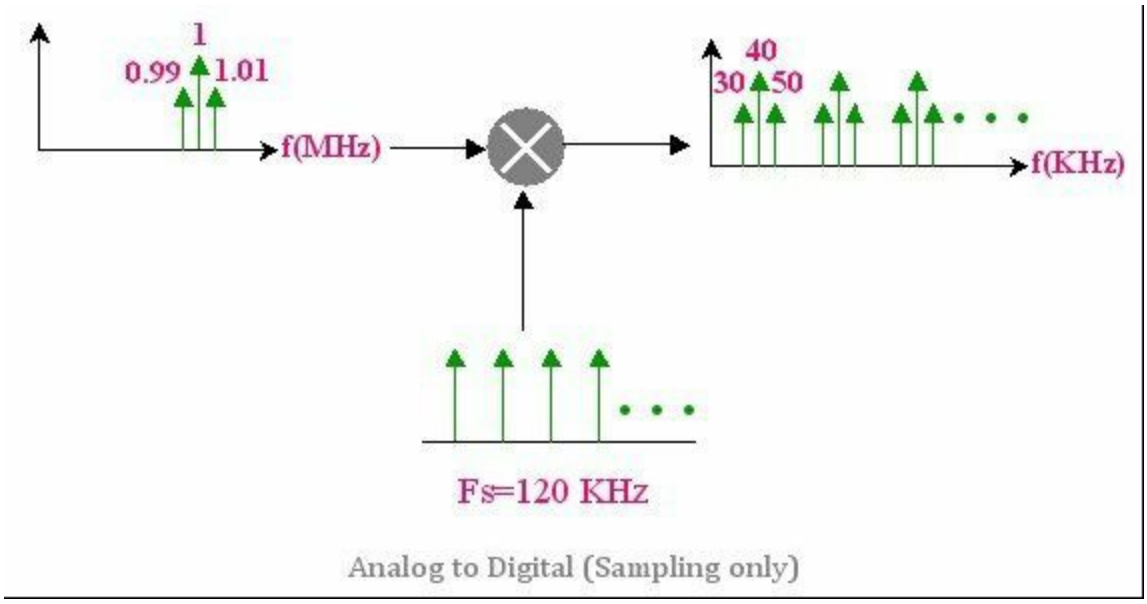
Let's set the sampling frequency to be F_S=120 KHz (which is 3 times higher than the minimum required sampling rate of 40 KHz or oversampling rate =3).

Now we can easily find the position of the spectral components in the sampled output by using the aliasing zone figure as given above. Since F_S=120 KHz, F_S/2 will be 60 KHz. So the zone 1 will be from 0 to 60 KHz, zone 2 -> 60-120 KHz and so on. The three spectral components at 0.99 MHz, 1 MHz and 1.01 MHz will fall at zone 17 (how ? 0.99 MHz/60 KHz = 16.5 , 1 MHz/60 KHz = 16.67 and 1.01 MHz/60 KHz = 16.83 , all figures approximating to 17). By the aliasing zone figure, zone 16 contains a copy of zone 17, zone 15 contains a copy of zone 16, zone 14 contains a copy of zone 15 and so on… Finally zone 1 contains the copy of zone 2 (Frequency reversal also exists at even zones). In effect, zone 1 contains a copy of zone 17. Since the original spectral components are at zone 17, which is an odd zone, zone 1 contains the copy of spectral components at zone 17 without

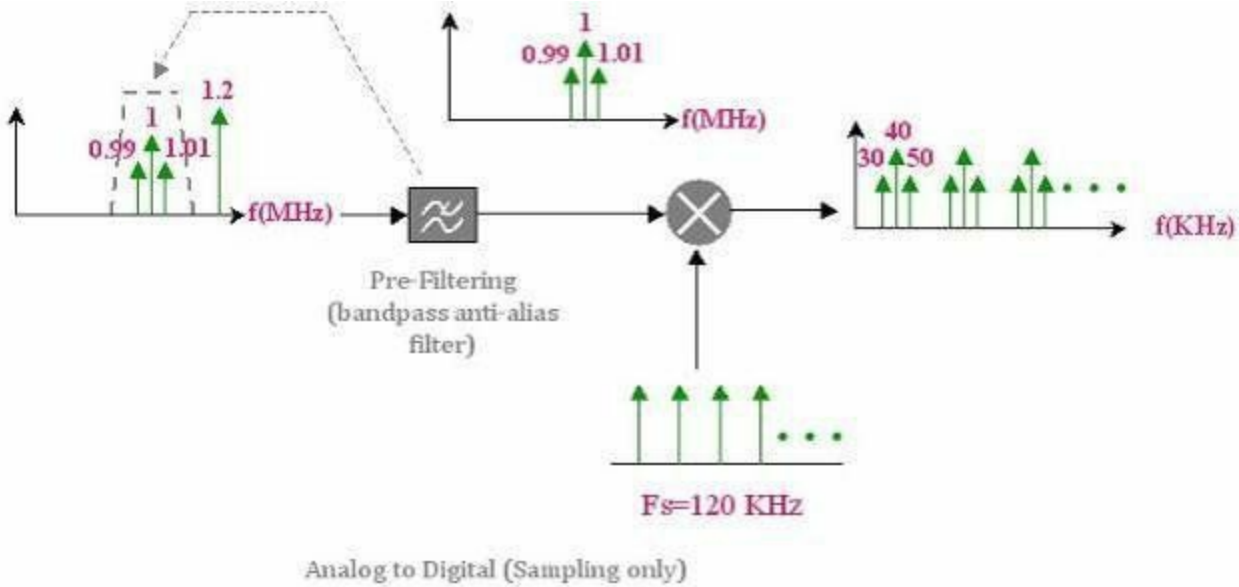
frequency reversal.

Since there is no frequency reversal, in zone 1 the three components will be at 30 KHz, 40 KHz and 50 KHz (You can easily figure this out).

This operation has down converted our signal from zone 17 to zone 1 without distorting the signal components. The down converted signal can be further processed by using a filter to select the baseband down converted components. Following figure illustrates the concept of bandpass sampling.



Consider the same AM signal with three components at 0.99 MHz, 1 MHz and 1.01 MHz. Now we also have an "unwanted" fourth component at 1.2 MHz along with the incoming signal. If we sample the signal at 120 KHz, it will cause aliasing (because the bandwidth of the entire signal is 1.2-0.99 = 0.21 MHz = 210 KHz and the sampling frequency of 120 KHz is below twice the bandwidth). In order to avoid anti-aliasing and to discard the unwanted component at 1.2 MHz, an anti-aliasing bandpass filter has to be used to select those desired component before performing the sampling operation at 120 KHz. This is also called "pre-filtering". The following figure illustrates this concept.



******

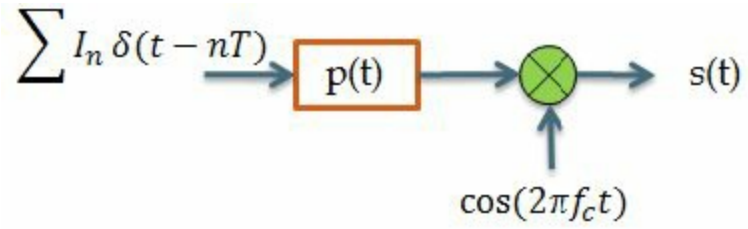
Back to Table of Contents

1.4 Oversampling, ADC – DAC Conversion, pulse shaping and Matched Filter

Digital communication systems involve conversion of digital data to analog form (with some modulation, coding stuffs etc...) at the transmitter side. The data transmitted through a channel is in analog form and it needs to be converted back to the digital domain by the process of Digital to Analog Conversion (DAC). Once the data is in digital domain, the receiver is at ease to process it to recover the transmitted data.

In Matlab one needs to emulate this situation where in the input data has to be represented in analog form in the transmitter and back to digital form at the receiver. Matlab is just a piece of software that runs on a digital machine (computer terminals), so it cannot really produce data in analog domain. To circumvent this problem, one has to use a process called oversampling.

Let's consider a digital communication system (shown below), where in, a train of input data (I_n = +/- 1) are shaped by a pulse shaping filter and modulated by a carrier F_c. This is a simple system implementing BPSK modulation. We will use rectangular pulse shaping filter (p(t)) for implementation in Matlab.



For illustration we try to transmit only one bit (I_1=1) through the system. To represent digital to analog conversion ("n" domain to "t" domain), we use oversampling by 4. Oversampling is a process in which additional "oversampling factor-1″ zeros are inserted in to the digital data.

For example, oversampling I_1=1 by a factor of 4 gives [1 0 0 0]. Similarly, oversampling the input [1 0 1 1] by a factor of 3 gives [1 0 0 0 0 0 1 0 0 1 0 0]. Following code snippet illustrates this.

```
overSampling_Factor=4;
Input_bit = 1;
Input_bit_os=upsample(Input_bit,overSampling_Factor); %oversampling by 4
%this is equivalent to performing digital to analog conversion in Tx side
```

The oversampled version of input is then passed through a pulse shaping filter (it can be a rectangular filter, root cosine filter, etc..,). A rectangular filter is implemented in the following code snippet.

```
%normalizing the pulse shape to have unit energy
pt = [ones(1,overSampling_Factor) 0 0 0 0 0 0]/sqrt(overSampling_Factor);
% impulse response of a rectangular pulse
%convolving the oversampled input with rectangular pulse
%The output of the convolution operation will be in the transmitter side
```

```
output_of_rect_filter = conv(Input_bit_os,pt);

stem(output_of_rect_filter);
title('Output of Rectangular Filter at Tx side')
xlabel('Samples')
ylabel('Amplitude')
```

******

Back to Table of Contents

1.5 Channel Capacity

How much data will a Channel/medium carry in one second or what is the data rate supported by the channel?

Let's discover the answer for this question in detail …

Shannon's Theorem:

Any discussion about the design of a communication system will be incomplete without mentioning Shannon's Theorem. Shannon's information theory tells us the amount of information a channel can carry. In other words it specifies the capacity of the channel. The theorem can be stated in simple terms as follows

* A given communication system has a maximum rate of information C known as the channel capacity.

* If the transmission information rate R is less than C, then the data transmission in the presense of noise can be made to happen with arbitrarily small error probabilities by using intelligent coding techniques.

* To get lower error probabilities, the encoder has to work on longer blocks of signal data. This entails longer delays and higher computational requirements.

The Shannon-Hartley theorem indicates that with sufficiently advanced coding techniques, transmission that nears the maximum channel capacity - is possible with arbitrarily small errors. One can intuitively reason that, for a given communication system, as the information rate increases, the number of errors per second will also increase.

Shannon – Hartley Equation:

Shannon-Hartley equation relates the maximum capacity (transmission bit rate) that can be achieved over a given channel with certain noise characteristics and bandwidth. For an AWGN the maximum capacity is given by (Check Appendix 1 and 2 for derivation of Shannon-Hartley equation for AWGN cannel)

$$C = B \log_2\left(1 + \frac{S}{N}\right)$$

Here C is the maximum capacity of the channel in bits/second otherwise called 'Shannon's capacity

limit' for the given channel, B is the bandwidth of the channel in Hertz, S is the signal power in Watts and N is the noise power, also in Watts. The ratio S/N is called Signal to Noise Ratio (SNR). It can be ascertained that the maximum rate at which we can transmit the information without any error, is limited by the bandwidth, the signal level, and the noise level.

It tells how many bits can be transmitted per second without errors over a channel of bandwidth B Hz, when the signal power is limited to S Watts and is exposed to Gaussian White (uncorrelated) Noise (N Watts) of additive nature.

Shannon's capacity limit is defined for the given channel. It is the fundamental maximum transmission capacity that can be achieved on a channel given any combination of any coding scheme, transmission or decoding scheme. It is the best performance limit that we hope to achieve for that channel.

The above expression for the channel capacity makes intuitive sense:

* Bandwidth limits how fast the information symbols can be sent over the given channel.

* The SNR ratio limits how much information we can squeeze in each transmitted symbols. Increasing SNR provides the transmitted symbols more robustness against noise. SNR is a function of signal quality, signal power and the characteristics of the channel. It is measured at the receiver's front end.

* To increase the information rate, the signal-to-noise ratio and the allocated bandwidth have to be traded against each other.

*. For no noise, S/N becomes infinite and so an infinite information rate is possible at a very small bandwidth.

Thus we may trade off bandwidth for SNR. However, as B tends to infinity, the channel capacity does not become infinite since, with an increase in bandwidth, the noise power also increases.

The Shannon's equation relies on two important concepts:

1) That, in principle, a trade-off between SNR and bandwidth is possible.
2) That, the Information capacity depends on both SNR and bandwidth

It is worth to mention two important works done by eminent scientists prior to Shannon's paper [Shannon1948].

Edward Amstrong's earlier work on Frequency Modulation (FM) is an excellent proof for showing that SNR and bandwidth can be traded off against each other. He demonstrated in 1936, that it was possible to increase the SNR of a communication system by using FM at the expense of allocating more bandwidth [Amstrong1938].

In 1903, W.M Miner in his patent (U. S. Patent 745,734), introduced the concept of increasing the capacity of transmission lines by using sampling and time division multiplexing techniques. In 1937, A.H Reeves in his French patent (French Patent 852,183, U.S Patent 2,272,070) extended the system by incorporating a quantizer, there by paving the way for the well-known technique of Pulse Coded Modulation (PCM). He realized that he would require more bandwidth than the traditional transmission methods and used additional repeaters at suitable intervals to combat the transmission noise. With the goal of minimizing the quantization noise, he used a quantizer with a large number of quantization levels. Reeves patent relies on two important facts:

1) Once can represent an analog signal (like speech) with arbitrary accuracy, by using sufficient frequency sampling, and quantizing each sample in to one of the sufficiently large pre-determined amplitude levels.

2) If the SNR is sufficiently large, then the quantized samples can be transmitted with arbitrarily small errors.

It is implicit from Reeve's patent – that an infinite amount of information can be transmitted on a noise free channel of arbitrarily small bandwidth. This links the information rate with SNR and bandwidth.

Please refer [Shannon1948] and [Shannon1949] for the actual proof by Shannon. A much simpler version of proof (I would rather call it an illustration) can be found at [WebScotts] (under the section "External Links" at the end of this book)

Unconstrained Shannon Limit for AWGN channel:

Some general characteristics of the Gaussian channel can be demonstrated. Consider that we are sending binary digits across an AWGN channel at a transmission rate equal to the channel capacity: R = C. If the average signal power is S, then the average energy per bit is E_b = S/C ((Joules per bit), since the bit duration is 1/C seconds. If the one sided noise power spectral density is $N_0/2$ Watts (normalized to 1 Ohm) per Hertz, then the total noise power is N_0B Watts. The Shannon-Hartley equation becomes

$$\frac{C}{B} = \log_2\left(1 + \frac{E_b}{N_0}\frac{C}{B}\right)$$

Rearranging the equation,

$$\frac{E_b}{N_0} = \frac{B}{C}\left(2^{\frac{C}{B}} \quad 1\right)$$

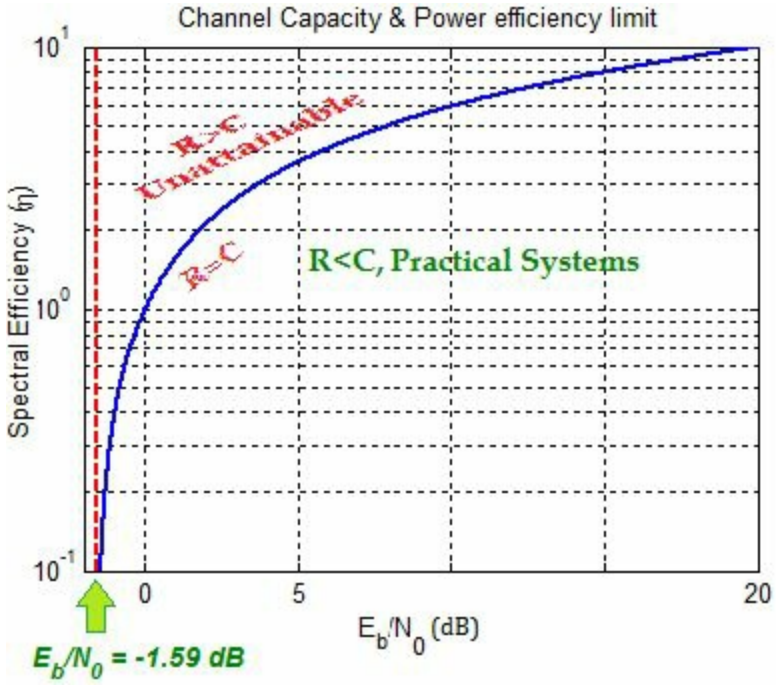
Letting C/B = η (the spectral efficiency in bits/seconds/Hz),

$$\frac{E_b}{N_0} = \frac{(2^{\eta} - 1)}{\eta}$$

A snippet Matlab code used to plot the above relationship is given below.

```
k =0.1:0.001:15;
E=(2.^k-1)./k;
semilogy(10*log10(E),k,'LineWidth',2.0);
xlabel('E_b/N_o (dB)');
ylabel('Spectral Efficiency (\eta)');
title('Channel Capacity & Power efficiency limit')
hold on;
grid on;
xlim([-2 20]);
ylim([0.1 10]);
yL = get(gca,'YLim');
line([-1.59 -1.59],yL,'Color','r','LineStyle','--','LineWidth',2.0);
```

In the graph that follows, the red dashed line in the graph represents the asymptote of E_b/N_0 as the bandwidth B approaches infinity. The asymptote is at E_b/N_0 =-1.59 dB (ln(2)). This value is called "Shannon's Limit" or specifically "Shannon's power efficiency limit".



Let's derive the Shannon's power efficiency limit. The asymptotic value (say 'x'), that we are seeking, is the value of E_b/N_0 as the spectral efficiency η approaches 0.

$$x = \lim_{n\to 0}\left(\frac{E_b}{N_0}\right) = \lim_{\eta\to 0}\left(\frac{2^\eta - 1}{\eta}\right)$$

Let $f(\eta) = 2^n - 1$ and $g(\eta) = \eta$. As $f(0) = g(0) = 0$ and the argument of the limit becomes indeterminate (0/0), L'Hospital'srule can be applied in this case. According to L'Hospital's rule, if $\lim_{x\to c} f(\eta)$ and $\lim_{x\to c} g(\eta)$ are both zero or are both $\pm\infty$, then

$$\lim_{x\to c}\left(\frac{f(x)}{g(x)}\right) = \lim_{x\to c}\left(\frac{f'(x)}{g'(x)}\right)$$

The next step boils down to finding the first derivative of $f(\eta)$ and $g(\eta)$.Expressing 2^n in natural logarithm,

$$2 = e^{\ln 2}$$

$$2^\eta = \left(e^{\ln 2}\right)^\eta = \left(e^{\eta \ln 2}\right)$$

Let $u = \eta \ln 2$ and $y = e^u$, then by chain rule of differentiation,

$$f'(\eta) = \frac{d2^\eta}{d\eta} = \frac{dy}{d\eta} = \frac{dy}{du}\frac{du}{d\eta} = e^u ln2 = ln2\ e^{\eta ln2}$$

$$x = \lim_{\eta\to 0}\left(\frac{f(\eta)}{g(\eta)}\right) = \lim_{\eta\to 0}\left(\frac{f'(\eta)}{g'(\eta)}\right) = \lim_{\eta\to 0}(ln2\ e^{\eta ln2}) = ln2 = 0.6931 = -1.59dB$$

Shannon's power efficiency limit does not depend on BER. Shannon's limit tells us the minimum possible E_b/N_o required for achieving an arbitrarily small probability of error as M->∞. (M is the

number of signaling levels for the modulation technique, for BPSK M=2, QPSK M=4 and so on...).

It gives the minimum possible E_b/N_o that satisfies the Shannon's-Hartley law. In other words, it gives the minimum possible E_b/N_o required to achieve maximum transmission capacity (R=C, where, R is the rate of transmission and C is the channel capacity). It will not specify at what BER you will get at that limit. It also will not specify which coding technique to use to achieve the limit. As the capacity is approached, the system complexity will increase drastically. So the aim of any system design is to achieve that limit. As an example, a class of codes called Low Density Parity Codes (LDPC) "near" the Shannon's limit but it cannot achieve it.

The Shannon limit derived above is called absolute Shannon power efficiency Limit. It is the limit of a band-limited system irrespective of modulation or coding scheme. This is also called "unconstrained" Shannon power efficiency Limit. If we select a particular modulation scheme or an encoding scheme, we can calculate the "constrained" Shannon limit for that scheme. We will see the generic form of Shannon Equation that applies to any channel and then later develop it to find the constrained capacities for AWGN channel.

In the next sections, generic forms of unconstrained Shannon equations are discussed for different types of generic channel models. These generic equations can be utilized to find the unconstrained capacity of a particular channel type – AWGN for example (check Appendix A1, A2 and A3 for details on how to extend the generic equation to a particular channel type).

Generic Form of Shannon Hartley Equation for a Discrete Memoryless Channel (DMC):

In this section, a very generic form of Shannon-Hartley equation for a band-limited discrete system is discussed. This generic form of equation can be used to obtain both absolute and constrained Shannon capacities for a particular channel.

All the following derivations are based on the works of the following references: [Ungerboeck1982], [Garrabrant1989], [Blahut1987], [Campanella1990] and [Wozencraft1965]

In a discrete memoryless channel, the input to the channel and the output of the channel are all treated as random variables taking values from a finite or infinite set of alphabets that are discrete in time. The memoryless condition indicates that the output of the channel at any time depends statistically only on the present input letter [Gallager1968, p 77].

Two variations to DMC exist with regard to the nature of values the input and output alphabets take.

1) Continuous input Continuous output Memoryless Channel (CCMC) – where the input and output alphabets can take continuous values

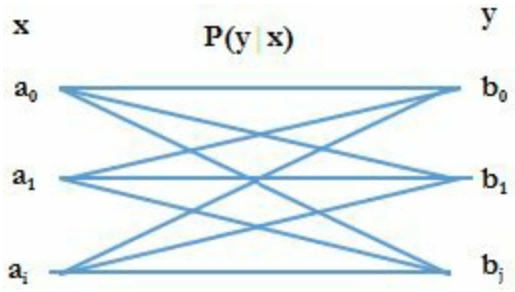
2) Discrete input Continuous output Memoryless Channel (DCMC) – where the input alphabets take value from finite set of value and the output values are continuous.

The above mentioned categories of channel models fall under category of vector channel models in which the input and output of the channel are treated as N-dimensional vectors. These models can be used to calculate the capacity of a specific M-ary signaling scheme. As an example, a QPSK signaling scheme is a two dimension signaling scheme. Each symbol of QPSK is limited in bandwidth and also in time, and it takes on values from a finite set of M-symbols. At the same time, due to the channel noise, the output at the sampling stage of the receiver can take any value. Thus, the capacity derived

for the N-dimensional DCMC model can be applied to find the capacity of the 2-dimensional QPSK signaling scheme on an AWGN channel.

A generic capacity formula for DMC is developed first. It is then extended to CCMC model. Then the capacity of M-ary signaling system is derived using the DCMC model.

Let's consider a generic Discrete Memoryless Channel where the input and the output alphabets are finite and are of finite duration. The above system can be diagrammatically represented as follows



In the above diagram, the input X and output Y are viewed as discrete random variables. The input event x can take "I" possible alphabets denoted by "a_i" and the output event y can take "J" possible values denoted by "b_j". The channel in between the output and the input is treated as a statistical system of transition probabilities given by P(Y|X).

$$x \in \{a_i;\ i = 1,2, \ldots, I\}$$

$$y \in \{b_j;\ j = 1,2, \ldots, J\}$$

The transition probabilities depend on the specific type of the channel. For example, if we use an AWGN channel, the transition probabilities follow a Gaussian PDF.

Each line on the above diagram gives the probability that a particular input symbol gets transitioned to the particular output symbol i.e, $P(y= b_j |x=a_i)$. For each line, the mutual information is given by

$$I_{X,Y}(a_i; b_j) = \log_2\left[\frac{P(x = a_i|y = b_j)}{P(x = a_i)}\right]$$

Taking the expectation of the above mutual information yields the average mutual information.

$$I(X;Y) = \sum_{i=1}^{M}\sum_{j=-\infty}^{\infty} P(x = a_i, y = b_j)\log_2\left[\frac{P(x = a_i|y = b_j)}{P(x = a_i)}\right] \quad (bits/symbol)$$

Noting the following relationship between conditional probability, marginal probability and joint probability

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

$$P(x,y) = P(y|x)p(x)$$

The average mutual information can be re-written in terms of a single conditional probability term.

$$I(X;Y) = \sum_{i=1}^{M}\sum_{j=-\infty}^{\infty} P(y = b_j|x = a_i)\,P(x = a_i)\log_2\left[\frac{P(y = b_j|x = a_i)}{P(y = b_j)}\right](bits/symbol)$$

Writing it in short form using the following abbreviation

$$P(x = a_i) = P(i)$$

$$P(y = b_j) = P(j)$$

The average mutual information is given by

$$I(X;Y) = \sum_{i=1}^{M} \sum_{j=-\infty}^{\infty} P(j|i)P(i)\log_2\left[\frac{P(j|i)}{P(j)}\right] \quad (bits/symbol)$$

The channel capacity C of the discrete memory less channel is defined as the maximum average mutual information. It is obtained by finding the set of input probabilities P(x=ai; i=0,1,…M-1) that maximizes the average mutual information I(X;Y).

$$C = \max_{P(i);\, i = 0,1,\ldots,M-1} I(X;Y)$$

$$C = \max_{P(i);\, i = 0,1,\ldots,M-1} \sum_{i=1}^{M} \sum_{j=-\infty}^{\infty} P(j|i)\,P(i)\log_2\left[\frac{P(j|i)}{P(j)}\right] \quad (bits/symbol)$$

Generic capacity equation for Continuous input Continuous output Memoryless Channel:

Here the input and the output are treated as an N-dimensional vectors (x_n and y_n) that takes continuous values from +∞ to -∞ a. The noise is also continuous and band-limited.

$$x_n \in \{-\infty, \infty\}$$

$$y_n \in \{-\infty, \infty\}$$

Since the input and the output are treated as vectors, we will denote them with a bar accent on the letters.

Here, an event is defined as the occurrence of specific set of values for the N-dimensional vector. The probability of a particular event is given by the product of probabilities of all the occurring values at a given instant.

$$P(\bar{x}) = \prod_{n=1}^{N} P(x_n)$$

$$P(\bar{y}) = \prod_{n=1}^{N} P(y_n)$$

The transition probability is given by

$$P(\bar{y}|\bar{x}) = \prod_{n=1}^{N} P(y_n|x_n)$$

The Capacity of the DMC model can be readily extended to CCMC model with N-dimensions as

$$C = \max_{P(\bar{x})} \int_{2N-Fold}^{-\infty,\infty} \cdots \int_{-\infty}^{\infty} P(\bar{y}|\bar{x})P(\bar{x})\log_2\left[\frac{P(\bar{y}|\bar{x})}{P(\bar{y})}\right] d\bar{x}d\bar{y} \quad (bits/symbol)$$

Appendix A1 and A2 give two different ways of deriving the Shannon's capacity equation for a CCMC AWGN channel.

Generic capacity equation for Discrete input Continuous output Memoryless Channel:

Remember that Shannon's limit is the theoretical limit for a given channel, no matter what type of coding scheme or transmission technique or decoding technique is employed. Shannon's limit tells us the minimum possible E_b/N_o required for achieving an arbitrarily small probability of error as M (the number of signaling levels) becomes infinite (i.e., input is also continuous). On the other hand, we cannot transmit with infinite signaling levels to achieve that capacity. In order to simplify transmission and to have a practical modulator, we restrict ourselves to transmitting with finite signaling levels by employing techniques like M-ary PSK and M-ary QAM. Therefore it is useful to know the theoretical limit for the given transmission scheme and any encoding/decoding technique. This type of capacity is called Constellation Constrained Capacity (as we are constraining the number of signaling points in the constellation- (check Appendix A3).).

This type of system with discretized input and continuous output is also called Discrete input Continuous output Memoryless Channel (DCMC).

A generic capacity formula for DCMC model is derived first. It is then extended to AWGN DCMC model (check Appendix A3).

For a DCMC system, any input symbol (x_m) takes on discrete values from the M-ary signal set (a_i) of M possible values (BPSK M=2, QPSK M=4,16-QAM M=16) The variable "N" represents the dimensionality (e.g., BPSK,QPSK and QAM are all two dimensional signals).Due to the noise, the output symbol (y) of the channel can take any value

$$x_{mn} \in a_i = \{a_1, a_2, \dots, a_M\}, \qquad n = 1,2, \dots, N$$

$$y_n \in b_j - \{-\infty, \infty\} \qquad n - 1,2, \dots, N$$

The transition probability of the channel is given by

$$P(\bar{y}|\bar{x}_m) = \prod_{n=1}^{N} P(y_n|x_{mn})$$

The Capacity of the DMC model can be readily extended to DCMC model with N-dimensions as given below..

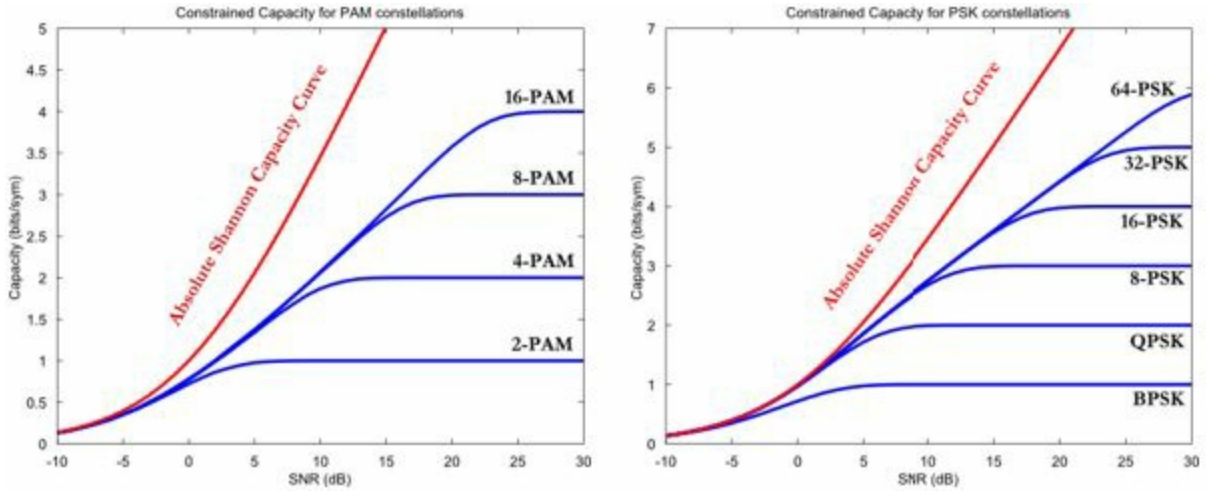
$$C = \max_{P(\bar{x}_m)} \sum_{m=1}^{M} \int_{N-Fold}^{-\infty} \dots \int_{-\infty}^{\infty} P(\bar{y}|\bar{x}_m) P(\bar{x}_m) \log_2 \left[\frac{P(\bar{y}|\bar{x}_m)}{P(\bar{y})} \right] d\bar{y} \quad (bits/symbol)$$

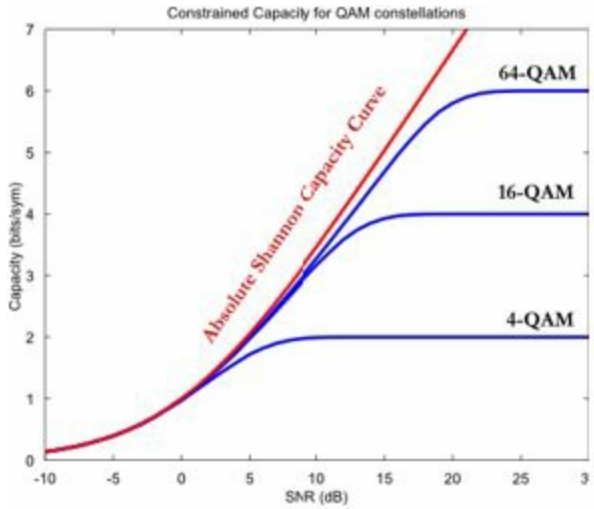
Let us compare this equation to the generic capacity equation for CCMC channel in the previous section. Since the input is discrete in this case, the 2N fold integral becomes N fold integral and the remaining N fold integral becomes a summation of N terms.

Check Appendix A3 for deriving the Shannon's capacity equation for M-ary Scheme (DCMC) over AWGN channel.

Constellation Constrained Capacity for AWGN channel for various modulation schemes:

The constellation constrained capacity for various modulation schemes over AWGN channel is given below. The following plots are obtained by modifying the Matlab code given at the external site links section [WebRobert]





Bandwidth-Power efficiency Trade-off:

There are two major communication resources namely transmitted power and channel bandwidth. In many communication systems, one of these resources is more precious than the other. It is important to strike a balance between these two resources without incurring any performance penalty. Generally all the communication systems are classified into two categories 1) Power limited systems and 2) Bandwidth limited systems.

Power-limited systems

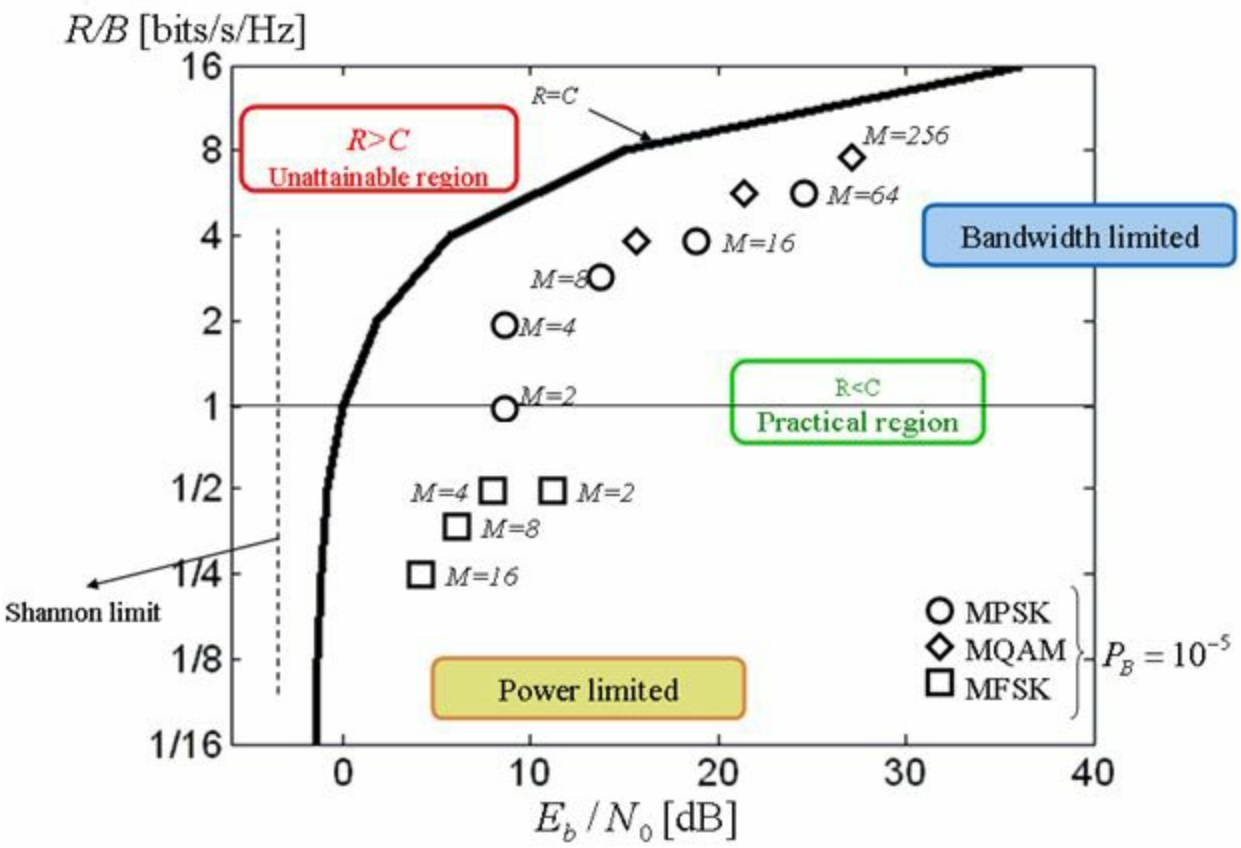
Saves power at the expense of bandwidth. For example, Frequency Shift Keying techniques like MFSK and CDMA - allocates more bandwidth for data transmission and operates with minimal power.

Bandwidth-limited systems

A bandwidth-limited system saves bandwidth at the expense of power. For example: spectrally efficient modulation schemes like MPSK and MQAM - limits the transmitted bandwidth at the expense of more transmission power

The following figure is for AWGN channel and it classifies various modulation techniques into power limited and bandwidth limited categories. It also partitions the graph into various regions

based on Shannon's limit. The following figure can be obtained by plotting the spectral efficiency curves for various modulation schemes and taking the points that give a bit error rate of 1E-5.



The R=C curve is the absolute Shannon limit for the AWGN channel. The region to the left of the curve where the transmission rate (R) is greater than the maximum unconstrained rate supported by the channel (C), is the unattainable region. It means that if we attempt to send data over channel at a rate greater than the Shannon capacity limit, the data errors at the receiver will be irrecoverable and we will lose the data completely.

The region to right of the R=C curve is the "practical or attainable" region. This is the region we have to operate at. Here the signaling rate R is less than the absolute capacity limit for the channel.

The individual points plotted for various modulation schemes are the constrained channel capacity points. These points give the maximum achievable capacity for a particular modulation scheme.

The modulation schemes are classified into either power limited or bandwidth limited scheme based on the spectral efficiency value (R/B).

******

Back to Table of Contents

1.6 Performance of Channel Codes

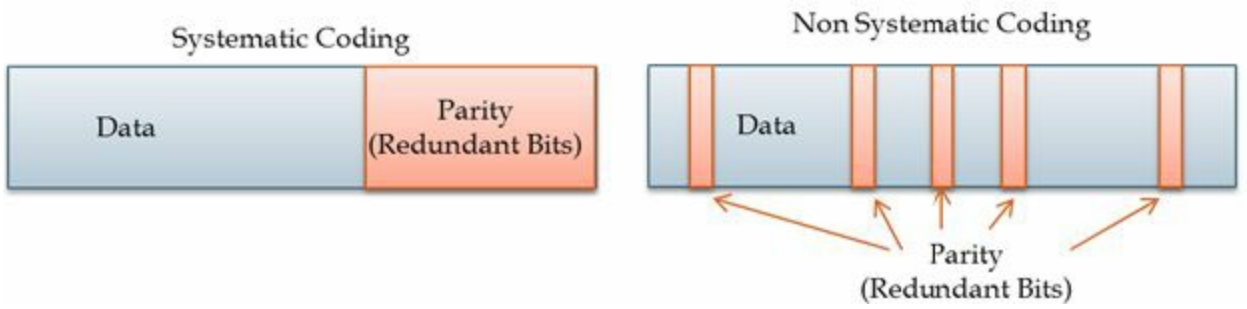
Encoding and Decoding:

Channel encoding and decoding techniques are used to make the transmitted data robust against

channel impairments. Even if the transmitted data gets corrupted by channel noise, it can be recovered by using channel coding techniques. Channel encoding adds redundant bits to the data. These redundant bits are used by the decoder in the receiver to mitigate the channel effects and recover the original data that was being sent.

There are two types of channel encoding techniques namely systematic encoding and non-systematic encoding. They both differ in terms of structure of the encoder output.

In systematic encoding, just by seeing the output of an encoder, we can separate the data and the redundant bits (also called parity bits). In the non-systematic encoding, the redundant bits get mixed with in the data bits.



Channel codes are also classified as “error detecting channel codes” and “error correcting channel codes”. Error detecting channel codes help the decoder detect any error in the transmission but they do not know how to correct those errors. Single Parity codes are excellent example for error detecting codes. On the other hand, error correcting codes assist the receiver in correcting those errors.

Channel Coding – Metrics:

Let’s look at the Channel Code Metrics in brief. There exists several types of Channel coding techniques and the task of choosing a specific code according to our need is made easier by some metrics. The metrics on which the channel codes are compared against each other are

* Coding Accuracy
* Coding Efficiency
* Coding Gain

Coding Accuracy:

A strong channel code usually recovers the data. But how close the recovered data matches with the original data (the information sent by the transmitter) is measured by Bit Error Rate (BER) probability.

$$P_{BER} = \frac{1}{K}\sum_{i=1}^{K} P_r(\widehat{W}_i \neq W_i)$$

Probability of bit error is the additive probability when the estimated symbol does not match the original transmitted symbol.

Coding Efficiency:

An efficient code has a relatively small number of encoded bits per data symbol. It is defined in terms

of code rate. Assume that each input symbol to the encoder is made up of K bits and the each output symbol from the encoder is of N bits in length (N=K + number of redundant bits).

Then the code rate is given by

$$R = \frac{K}{N}$$

More is the code rate more is the efficiency. It is impossible to achieve a code rate of unity, since the decoder estimates depend on the redundant bits. When you decrease the number of redundant bits, the number of errors per symbol it can correct and/or detect also decreases.

So you cannot design a code that arbitrarily reduces the number of redundant bits (otherwise increase the code rate) since it affects the error correcting or detecting capability of the code.

So a trade-off does exist between the code efficiency and the error correcting or error detecting capability of the code. Therefore, whenever we talk about the code rate, the error correcting or detecting capability of the codes also comes with it.

Error Correction and Detection:

There exist codes that can detect errors in a symbol (at the decoder side) but do not do anything if an error occurs. It is the responsibility of the next block to correct the errors. Some codes can only correct errors and some can do both detection and correction simultaneously.

All these capabilities can be measured by a parameter called Hamming Distance. (For more on Euclidean distance – another type of distance metric – see the later chapters)

Consider a scenario in which the only possible data I can transmit are the vowels A, E, I and O (say, I do not like transmitting the vowel 'U'). So to transmit these four vowels I need two bits (two bits can represent 2^2=4 symbols or 4 vowels in our case). My encoder accepts the input in terms of two bits or one symbol at a time and outputs three bits or one output symbol. Out of the three bits in the output, the first two bits represent my original input vowels and the last bit is the redundant bit which is added to satisfy even parity (number of '1's in the output symbol is always even). This implies that the encoder uses even parity mechanism. This is a type of channel code. The following table summarizes the coding scheme described above

Input Vowel	In Binary	Output Symbol	Hamming Weight
A	00	000	0
E	01	011	2
I	10	101	2
O	11	110	2

From the above table, the Hamming Distance for our coding scheme can be calculated as

Hamming Distance = min (Hamming Weights excluding '0′ weight) = 2

The output symbols are called codewords which hold the secret of detecting or correcting the error at the receiver.

Consider that I employ five people to carry each CODEWORDS (output symbols) mentioned above and ask them to deliver it to my client. When these five people reach my client at the destination, they

have to tell the codeword. The first fellow tells 110 and my client knows that it is correct codeword - since the number of '1' in the codeword are even (This may also be incorrect if two bits get corrupted). The second fellow tells 010 and the client knows that the vowel is corrupted (since the number of ones in the received codeword is odd) but doesn't know how to determine the correct word.

So my codeword mechanism can detect 1 bit error and cannot correct it.

Look at the table again. The first codeword with all zeros is called the "Zero Code Word". The last column in the table specifies a parameter called Hamming Weight for each codeword. It is the number of places the codeword differs from the Zero Code Word. For example the second codeword is '011' and it differs in '2' places when compared to the Zero Code Word.

Hamming Distance is the minimum of all the Hamming weights. In this table since all the Hamming weights are '2', the Hamming distance is also '2' (do not consider the hamming weight for the Zero Code Word while doing so).

Hamming distance speaks volumes about the error correcting or detecting capability of a code.

The Distance Rule:

If a code can detect -'t' number of errors, then its Minimum Hamming distance must be t+1.

If a code can correct -'t' number of errors, then its Minimum Hamming distance must be 2t+1.

Let's apply these rules to the even parity coding scheme described above. The minimum Hamming distance of the code is '2'. That is t+1 = 2 , which implies t=1, so the code can detect '1' bit error.

Consider a coding scheme which has a minimum Hamming distance of '5'. It can detect 4 bit errors and can correct any 2 bit errors.

Coding Gain:

Coding gain for a coded system, usually expressed in dB, is the difference in Signal-to-Noise-Ratio (SNR) required between the coded and uncoded systems achieving the same probability of error.

Let's say I have an uncoded system (it is not using any Error Correction Codes). It gives me a BER of 10^-5 for an SNR of 3 dB. Now I introduce a Channel coding technique in the system and it achieves the Same BER of 10^-5 at an SNR of 1.4 dB. The difference in SNR for achieving the same BER (1.6 dB in our case) is called coding gain. This means that for the same performance benchmark, the coded system required very low power to operate when compared to the uncoded system.

The coding gain increases with increasing minimum Hamming distance, which increases the code length which in turn decreases the code rate. But remember that our code may be very efficient yet it can never cross the Shannon's Limit ….

Classification of Channel Codes

The channel codes can be classified into two broad categories.

1) Block Codes
2) Convolutional codes.

There are many differences between block codes and convolutional codes. Block codes are based rigorously on finite field arithmetic and abstract algebra. They can be used to either detect or correct errors. Block codes accept a block of 'k' information bits and produce a block of 'n' coded bits. By predetermined rules, (n-k) redundant bits are added to the 'k' information bits to form the 'n' coded bits. Commonly, these codes are referred to as (n, k) block codes. Some of the commonly used block codes are Hamming codes, Golay codes, BCH codes, and Reed Solomon codes (uses non-binary symbols).

Convolutional codes are one of the most widely used channel codes in practical communication systems. These codes are developed with a separate strong mathematical structure and are primarily used for real time error correction. Convolutional codes convert the entire data stream into one single codeword. The encoded bits depend not only on the current 'k' input bits but also on past input bits. The main decoding strategy for convolutional codes is based on the widely used Viterbi algorithm.

As a result of the wide acceptance of convolutional codes, there have been many advances to extend and improve this basic coding scheme. This advancement resulted in two new coding schemes, namely, trellis coded modulation (TCM) and turbo codes.

TCM adds redundancy by combining coding and modulation into a single operation (as the name implies). The unique advantage of TCM is that there is no reduction in data rate or expansion in bandwidth as required by most of the other coding schemes. Recently, a near channel capacity error correcting code called turbo code was introduced. This error correcting code is able to transmit information across the channel with arbitrarly low (approaching zero) bit error rate. This code is a parallel concatenation of two component convolutional codes separated by a random interleaver.

Apart from classifying the channel coding techniques into block codes and convolutional codes, it is also important to understand the decoding nature of the receiver. The channel decoder at the receiver can act on hard bits or soft bits and thus the decoding technique can be classified as hard decision or soft decision decoding respectively. The next two sections provides the basis for understanding hard decision and soft decision decoding techniques.

******

Back to Table of Contents

1.7 Distances: Hamming Vs. Euclidean

Hamming Distance:

The concept of Hamming distance was discussed in the previous section. Let's do a recap on the concept of Hamming distance:

Hamming distance between two codewords is the count of number of bits that are different in the two codewords. Hamming weight of a non-zero codeword is the number of '1's in the codeword, in other words, it is the hamming distance between the codeword and the ALL-ZERO codeword

Minimum Hamming Distance (MHD) of a block code essentially tells us the error correcting

capability of the code. MHD of a coding scheme is the lowest hamming weight among all the codewords in a coding scheme.

Example:

Consider a (3, 1) even parity coding. Here each codeword is of length 3 bits and has one parity bit. Let's call the bits – x0, x1 and x2. Here x0 and x1 are the information bits and x2 is the parity bit. The coding rule is given by

$$x_2 = x_0 \oplus x_1$$

Then the list of all codewords is given by

x_0	x_1	x_2	Hamming Weight
0	0	0	0
0	1	1	2
1	0	1	2
1	1	0	2

The error detection and correction capability rules are repeated here for convenience.

Rules for Error Detection and Correction:

If a code can detect "t" number of errors, then its Minimum Hamming distance must be t+1.

If a code can correct "t" number of errors, then its Minimum Hamming distance must be 2t+1.

Applying these rules to the above mentioned example: Minimum Hamming Distance= d_{min} = lowest Hamming weight -> which is '2' (leave the ALL zero code word). As we discussed earlier, since=> d_{min} = t+1 =2, => t=1, it is able to detect single bit error.

But this d_{min} will not satisfy the equation 2t+1, (because, if 2t+1=2 => t=0.5 => meaning that it will be able to correct 1/2 bit errors - which is absurd). So we may conclude that this parity coding scheme can detect single error but cannot correct even a single error.

In order to correct the corrupted bits at the receiver, the following two variables must be known at the receiver.

1) Position of the bits corrupted.
2) The original value of the bits.

The even parity decoding scheme that was described above cannot provide this information. It can only detect a single bit error. Even if all the bits are flipped (3 bit errors), this coding scheme will still report it as a single bit error.

Thus, d_{min} plays an important role in determining the error correcting capability of a code. An encoder is an optimum minimum distance encoder, if its minimum distance is equal or superior to that of any encoder of the same code rate.

During code design, designers won't always use the Hamming distance as a metric for the design. There exists another metric called Euclidean Distance.

Euclidean Distance:

Hamming distance is not always suitable for code design. In general, if we use a soft decision decoder at the receiver, Euclidean metric is used as design criteria and the coding schemes are chosen that maximize the minimum Euclidean distance.

Also if the channel is assumed to be Gaussian with high SNR, Euclidean distance dominates the error bound. However, Euclidean distance is not used for Raleigh Fading channels and hard decision decoding, instead, Hamming distance is used as a criterion in such scenarios.

Both Hamming distance and Euclidean distance can be summarized using a general formula called MinKowsky equation given by

$$d(A,B) = \left[\sum_{i=1}^{n} w_i[a_i - b_i]^P\right]^{\frac{1}{P}}$$

$$if, P = 1 \Rightarrow Manhattan\ Distance$$
$$P = 2 \Rightarrow Euclidean\ Distance$$
$$P = 1\ and\ a_i, b_i \in \{0,1\} \Rightarrow Hamming\ Distance$$

******

Back to Table of Contents

1.8 Hard and Soft Decision Decoding

To understand hard and soft decision decoding, we have to begin with the sampling and quantization process in the receiver.

Quantization levels:

In a receiver, the received signal is in analog form. The analog received signal is converted into digital domain by sampling, quantization and coding process.

Consider a BPSK receiver, where the only two possible symbols are '0' and '1'. The symbol '0' is transmitted using a sine wave of '0' phase shift and '1' is transmitted with a phase shift of 180°.

In the hard decision receiver, the sampler in the receiver, samples the received symbol at a sampling instant and the quantizer quantizes the sampled value to either '0' or '1'. This is the simplest form of quantization where only two levels are used. The hard decision decoder simply decides whether the received bit is a one or zero.

In a soft decision receiver, the quantizer is a multilevel quantizer. It quantizes the sampled symbol into multiple levels. For example, the BPSK symbols can be quantized into eight levels (each bit represented by 3 bits) as shown in the next figure. This provides more information and more confidence for the decision process that follows. Now the zeros and ones can be classified into eight levels of intensity – staring from weakest to strongest.

Furthermore, the coder that follows the quantizer can adopt any of the coding schemes for representing each bit– two's complement, offset binary and signed magnitude. The onus lies on the designer to interpret the representation properly. A sample table, illustrating how the soft bits can be

interpreted in each of the commonly used binary representations, is shown next [QualcommAN1652].

Codewords			
Offset binary	Signed Magnitude	2's complement	Interpretation
000	011	100	Strongest 0
001	010	101	Strong 0
010	001	110	Weak 0
011	000	111	Weakest 0
100	100	000	Weakest 1
101	101	001	Weak 1
110	110	010	Strong 1
111	111	011	Strongest 1

In a Gaussian Channel, the 8-level quantization gives better performance when compared to 2-level quantization. The 8-level quantization improves the required SNR by approximately 2 dB. Infinite quantization level improves the SNR by 2.2 dB. Therefore, the 8-level quantization for this particular case is just 0.2 dB short compared to infinite quantization scheme [Heller1971].

Apart from the difference in quantization levels, the distance metric used by the decoders also plays a vital role in hard & soft decision decoding.

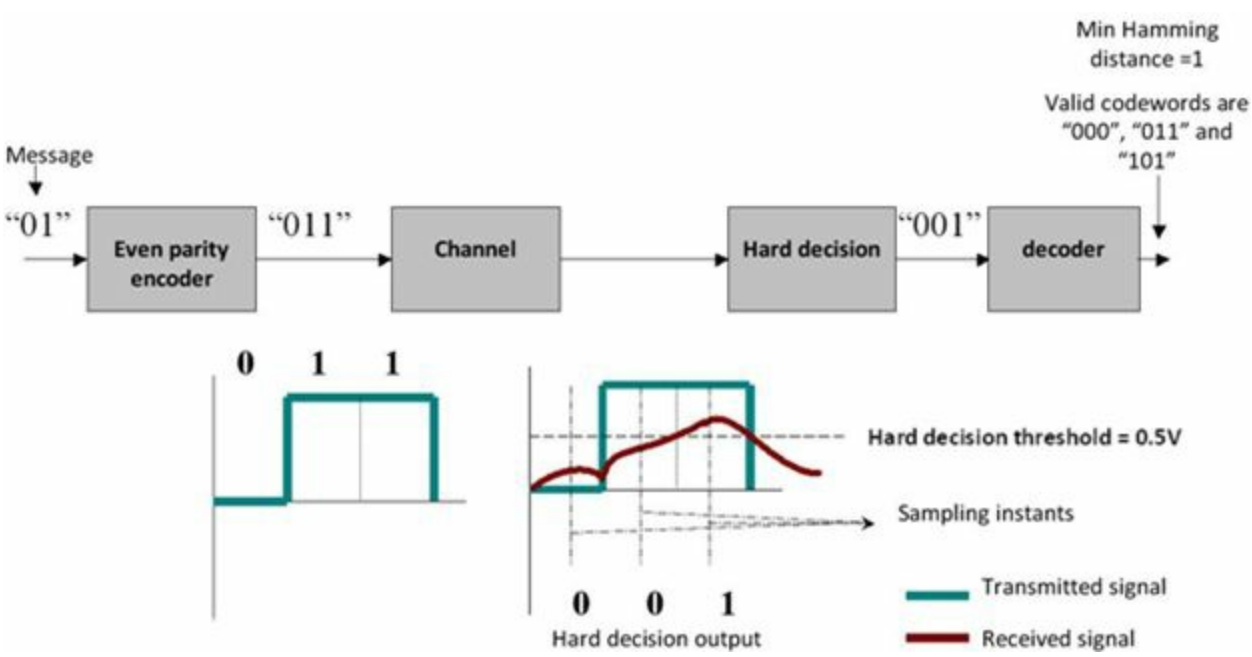
Distance Metric and Decoding:

Consider a simple even parity encoding scheme as given in the following table. The set of all possible codewords generated by the encoder are 000,011,101 and 110. Let us say we want to transmit the message "01" through a communication channel.

Input Bit 1	Input Bit 2	Parity Bit added by the encoder	Codeword generated
0	0	0	000
0	1	1	011
1	0	1	101
1	1	0	110

Hard decision decoding:

Case 1: Assume that our communication model consists of a parity encoder, communication channel (attenuates the data randomly) and a hard decision decoder. The message bits "01" are applied to the parity encoder and we get "011″ as the output codeword.



The output codeword "011" is transmitted through the channel. "0" is transmitted as "0 Volt" and "1" as "1 Volt". The channel attenuates the signal that is being transmitted and the receiver sees a distorted waveform (the signal depicted in red color in the figure above). The hard decision decoder makes a decision based on the threshold voltage. In our case the threshold voltage is chosen as 0.5 Volt (midway between "0" and "1" Volt). At each sampling instant in the receiver (as shown in the figure above) the hard decision detector determines the state of the bit to be "0" if the voltage level falls below the threshold and "1" if the voltage level is above the threshold. Therefore, the output of the hard decision block is "001". Perhaps this "001" output is not a valid codeword (compare this with all possible codewords given in the table above), which implies that the message bits cannot be recovered properly. The decoder compares the output of the hard decision block with all possible codewords and computes the minimum Hamming distance for each case (as illustrated in the following table).

Comparing All possible codewords and Hard decision output		
All possible codewords	**Hard decision output**	**Hamming distance**
000	001	1
011	001	1
101	001	1
110	001	3

The decoder's job is to choose a valid codeword which has the minimum Hamming distance. In our case, the minimum Hamming distance is "1" and there are 3 valid codewords with this distance. The decoder may choose any of the three possibilities and the probability of getting the correct codeword ("001" – this is what we transmitted) is always 1/3. So when the hard decision decoding is employed the probability of recovering our data (in this particular case) is 1/3. Let's see what "Soft decision decoding" offers …

Soft Decision Decoding

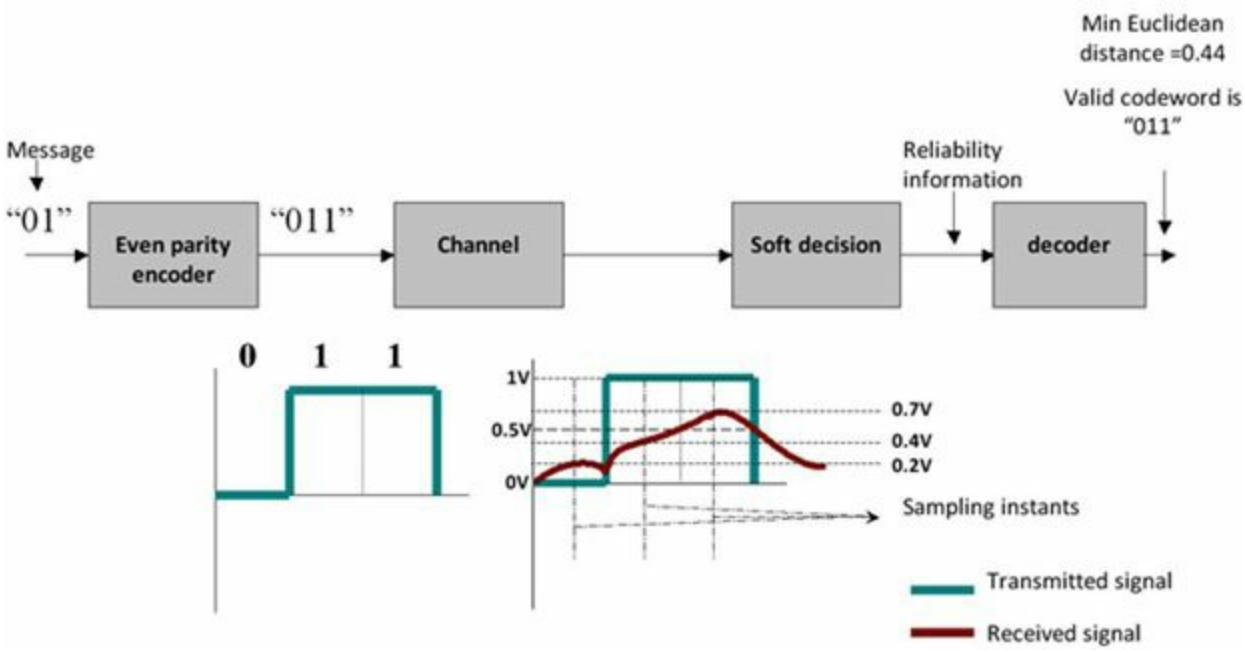
The difference between hard and soft decision decoder is summarized as follows

* In Hard decision decoding, the received codeword is compared with all possible codewords and the codeword which gives the minimum Hamming distance is selected

* In Soft decision decoding, the received codeword is compared with all possible codewords and the codeword which gives the minimum Euclidean distance is selected. Thus the soft decision decoding improves the decision making process by supplying additional reliability information (calculated Euclidean distance or calculated log-likelihood ratio)

For the same encoder and channel combination let us see the effect of replacing the hard decision block with a soft decision block.

Voltage levels of the received signal at each sampling instant are shown in the figure. The soft decision block calculates the Euclidean distance between the received signal and all possible codewords.



The next table illustrates how Euclidean distance can be calculated for a given set of codewords.

Valid codewords	Voltage levels at each sampling instant of received waveform	Euclidean distance calculation	Euclidean distance
0 0 0	0.2V 0.4V 0.7V	(0-0.2)2+ (0-0.4)2+	0.69
(0V 0V 0V)			
0 1 1	0.2V 0.4V 0.7V	(0-0.2)2+ (1-0.4)2+ (1-0.7)2	0.49
(0V 1V 1V)			
1 0 1	0.2V 0.4V 0.7V	(1-0.2)2+ (0-0.4)2+ (1-0.7)2	0.89
(1V 0V 1V)			
1 1 0	0.2V 0.4V 0.7V	(1-0.2)2+ (1-0.4)2+ (0-0.7)2	1.49
(1V 1V 0V)			

The minimum Euclidean distance is "0.49" corresponding to "0 1 1" codeword (which is what we transmitted). The decoder selects this codeword as the output. Even though the parity encoder cannot correct errors, the soft decision scheme helped in recovering the data in this case. This fact delineates the improvement that will be seen when this soft decision scheme is used in combination with forward error correcting (FEC) schemes like convolution codes, LDPC etc.

From this illustration we can understand that the soft decision decoders use the whole information (voltage levels in this case) in the process of decision making, whereas the hard decision decoders doe not fully utilize the information available in the received signal (evident from calculating Hamming distance just by comparing the signal level with the threshold whereby neglecting the actual voltage levels).

Note: This is just to illustrate the concept of soft decision and hard decision decoding. Prudent souls will be quick enough to find that the parity code example will fail for other voltage levels (e.g.: 0.2V, 0.4 V and 0.6V). This is because the parity encoders are not capable of correcting errors but are capable of detecting single bit errors. Nevertheless, the soft decision decoding is very helpful in improving the performance of a communication system.

Soft decision decoding scheme is often realized using Viterbi decoders. Such decoders utilize Soft Output Viterbi Algorithm (SOVA) which takes into account the priori probabilities of the input symbols, thereby producing a soft output indicating the reliability of the decision.

******

Back to Table of Contents

1.9 Maximum Likelihood Decoding

Having covered the techniques of hard and soft decision decoding, it's time to illustrate the most important concept of Maximum Likelihood Decoding.

Maximum Likelihood Decoding:

Consider a set of possible codewords (let us say 'Y') generated by an encoder in the transmitter side. We pick one codeword out of this set (call it 'y') and transmit it via a Binary Symmetric Channel (BSC) with probability of error p (refer Chapter 1 section 1.1 for definition on BSC). At the receiver side we receive the distorted version of 'y' (call this erroneous codeword 'x').

Maximum Likelihood Decoding chooses one codeword from the set 'Y' (the list of all possible codewords) which maximizes the following probability.

$$\mathbb{P}(y \text{ sent } | x \text{ received})$$

Meaning that the receiver computes probabilities $P(y_1,x)$, $P(y_2,x)$, $P(y_3,x)$,…,$P(y_n,x)$ and it chooses a codeword (y) that gives the maximum probability.

In practice we don't know Y (at the receiver) but we know 'x' (the actual received symbol/sample). So, how to compute the probability? Maximum Likelihood Estimation (MLE) comes to our rescue. For a detailed explanation on MLE – refer [WebPurcell]

The aim of maximum likelihood estimation is to find the parameter value(s) that makes the observed data most likely. Understanding the difference between prediction and estimation is important at this point.

Estimation differs from prediction in the following way.

In estimation problems, likelihood of the parameters is estimated based on given data/observation vector. In prediction problems, probability is used as a measure to predict the outcome from known parameters of a model.

Examples for "Prediction" and "Estimation":

1) Probability of getting a "Head" in a single toss of a fair coin is 0.5. The coin is tossed 100 times in a row. Prediction helps in predicting the outcome (head or tail) of the 101th toss based on the probability.

2) A coin is tossed 100 times and the data (head or tail information) is recorded. Assuming the event follows Binomial distribution model, estimation helps in determining the probability of the event. The actual probability may or may not be 0.5.

Maximum Likelihood Estimation estimates the conditional probability based on the observed data (received data – x) and an assumed model.

Example of Maximum Likelihood Decoding:

Let y=11001001 and x=10011001. Assuming Binomial distribution model for the event with probability of error 0.1 (i.e. the reliability of the BSC is 1-p = 0.9), the Hamming distance between codewords is y-x = 2. For binomial channel model,

$$\mathbb{P}(y\ sent\ |x\ received) = (1-p)^{n-d}p^{d}$$

Here, d is the Hamming distance between the received and the sent codewords, n is the number of bits sent, p is the error probability of the BSC and 1-p is the reliability of BSC

Substituting d=2, n=8 and p=0.1, then,

$$\mathbb{P}(y\ sent\ |x\ received) = 0.005314$$

Note: Here, Hamming distance is used to compute the probability. So the decoding can be called as "minimum distance decoding" (which minimizes the Hamming distance) or "maximum likelihood decoding". Euclidean distance may also be used to compute the above mentioned conditional probability.

As mentioned earlier, in practice y is not known at the receiver. Let us see how to estimate P(y sent / x received), when y is unknown, based on the binomial model.

Since the receiver is unaware of the particular y corresponding to the x received, the receiver computes P(y sent / x received) for each codeword in the set Y. The "y" which gives the maximum probability is concluded as the codeword that was sent. Let us dig little deeper into Maximum Likelihood Estimation, the idea behind Maximum Likelihood Decoding.

Maximum Likelihood Estimation:

Maximum Likelihood estimation (MLE) is an important tool in determining the actual probabilities of the assumed model of communication- based on the observed data.

In reality, a communication channel can be quite complex and a model becomes necessary to simplify calculations at decoder side. The model should closely approximate the complex communication channel. There exist a myriad of standard statistical models that can be employed for this task;

Gaussian, Binomial, Exponential, Geometric, Poisson, etc., A standard communication model is chosen based on empirical data.

Each model mentioned above has unique parameters that characterize them. Determination of these parameters for the chosen model is necessary to make them closely model the communication channel at hand.

Suppose a binomial model is chosen (based on observation of data) for the error events over a particular channel, it is essential to determine the probability (p) of the binomial model.

If a Gaussian model (normal distribution!!!) is chosen for a particular channel then estimating μ (mean) and σ^2 (variance) are necessary so that they can be applied while computing the conditional probability of p(y received | x sent)

Similarly estimating λ (lambda) is a necessity for a Poisson distribution model.

Maximum likelihood estimation is a method to determine these unknown parameters associated with the corresponding chosen models of the communication channel.

An Example for MLE:

The following data is presented based on the observations over a Binary Symmetric Channel (BSC) (p=0.5) modeled as a binomially distributed model.

90 codewords (each of them 10 bits wide) are transmitted over the BSC channel, of which 410 bits in total are received erroneously. Let us estimate the actual probability of success for this model.

The BSC channel is assumed to have p=0.5, meaning that the probability of error for ’0′ and ’1′ are equal and the probability of success is also 0.5.

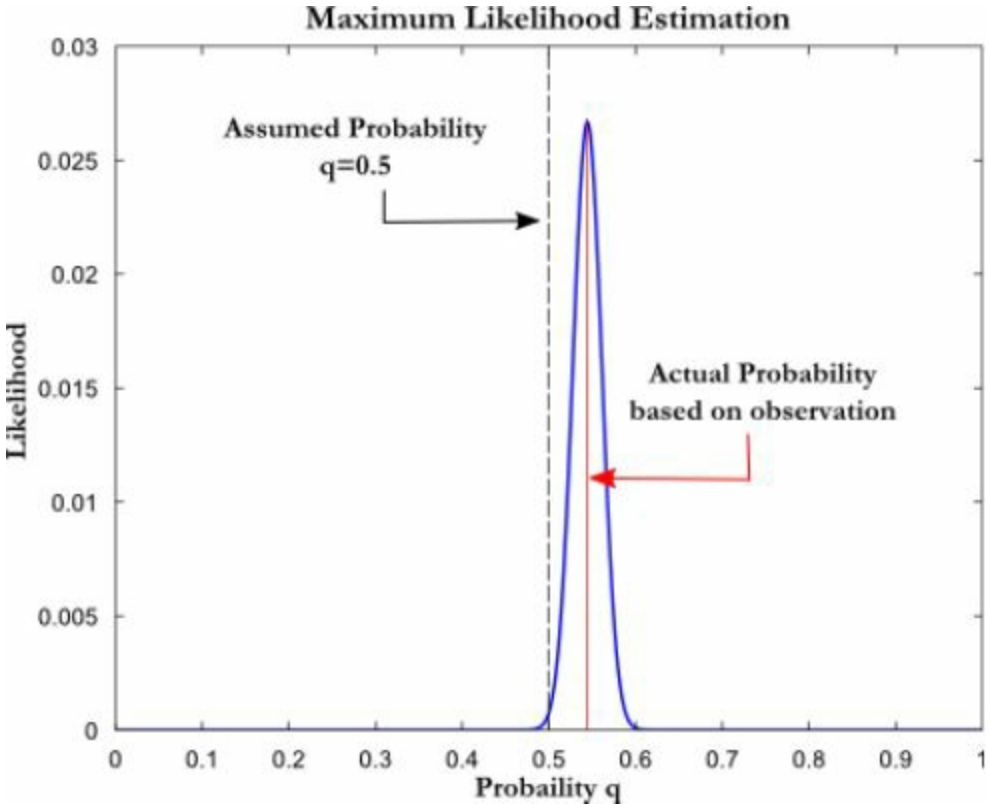
Matlab Code:

```
% Maximum likelihood Estimation
d=410; %Number of bits in error
n=90*10; %Total number of bits sent
k=n-d; %Number of Bits NOT in error
q=0:0.002:1; %range of success probability to test likelihood
y=binopdf(k,n,q); % assuming binomial distribution
plot(q,y);
xlabel('Probaility q');
ylabel('Likelihood');
title('Maximum Likelihood Estimation');
[maxY,maxIndex]=max(y); % Finding the Max and its index
disp(sprintf('MLE of q is %f',q(maxIndex))) %print the probability corresponding to the max(y)
```

Output :

MLE of q is 0.544000

Therefore the probability of success for this BSC is q= 0.544 whereas the probability of error is p=1-q=0.456



******

Back to Table of Contents

Chapter 2: Channel Coding

2.1 Hamming Codes - How it works

Introduction:

Hamming Codes fall under the category of linear Block codes of Forward Error Correcting (FEC) codes. Let us consider a (7, 4) hamming code. The notation (7, 4) indicates that the codewords are of length 7 bits. Out of these 7 bits, 4 bits are the original message and remaining 3 bits are added for detecting and correcting errors. These remaining 3 bits are called redundant bits.

The structure can be indicated as follows:

4 message bits + 3 redundant bits => 7 bit Hamming code.

Generally the linear block codes can be represented in two forms. One is called Systematic form and other is called non-Systematic form. In Systematic Coding, the redundant bits are calculated from the message bits and both are concatenated side by side. Just by looking at the codeword you can identify the message portion and the redundant portion. For example if I have the following (7, 4) code, let's say 1010111 (this is not a Hamming code, just used for illustration), and I say that it is represented in a systematic form. Then you can easily identify the first 4 bits 1010 as the original message bits and the remaining bits 111 as the redundant bits. In non-systematic codes, you cannot identify them unless you know what type of coding method was used. It means that the positions of the original message bits and the redundant bits are mixed.

Hamming code can also be represented in two forms: systematic and non-systematic form. We will look at the non-systematic form, since it is easier to illustrate the concept behind Hamming codes.

Construction of Hamming codes:

Consider transmitting 4 data bits and these data bits are represented by letter 'D'. We are going to find the 3 redundant bits (represented by letter 'P') using Hamming code algorithm and form the 7 bit Hamming code. The codewords made in this way is called (7, 4) Hamming code which is a very basic code.

Let the codeword bits be represented by "$D_7\ D_6\ D_5\ P_4\ D_3\ P_2\ P_1$". Here D_7, D_6, D_5 and D_3 are the message bits and P_4, P_2, and P_1 are the parity or redundant bits. The parity bits are calculated using the following equations. Here '+' sign indicates modulo-2 addition or XOR operation.

$$P_4 = D_7+D_6+D_5$$
$$P_2 = D_7+D_6+D_3$$
$$P_1 = D_7+D_5+D_3$$

The following table illustrates how to calculate parity bits for the above coding scheme.

7	6	5	4	3	2	1	How to calculate parity bits
D_7	D_6	D_5	P_4	D_3	P_2	P_1	This represents the full codeword
D_7	D_6	D_5	P_4				P_4 – Even parity of $D_7\ D_6\ D_5$
D_7	D_6			D_3	P_2		P_2 – Even parity of $D_7\ D_6\ D_3$
D_7		D_5		D_3		P_1	P_1 – Even parity of $D_7\ D_5\ D_3$

Let us attempt to find the Hamming code for the message bits "1101". The message "1101" will be sent as "1100110" using Hamming coding algorithm as follows. Here the data bits and the parity bits in the codeword are mixed in position and so it is a non-systematic code.

D_7	D_6	D_5	P_4	D_3	P_2	P_1	How to calculate parity bits
1	1	0	0	1	1	0	This represents the full codeword
1	1	0	0				P4 – Even parity of 1 1 0 = 0
1	1			1	1		P2 – Even parity of 1 1 1 = 1
1		0		1		0	P1 – Even parity of 1 0 1 = 0

The 4-bit message is converted into 7 bit codeword. This means, out of 128 combinations (2^7 =128) only 16 combinations are valid codewords. At the decoder side, if we receive these valid codewords then there is no error. If any of the other combinations (apart from the valid codewords) are received then it is an error. The minimum Hamming distance of the given Hamming code is 3, this indicates that

the Hamming code can detect 2 bit errors or it can correct single bit error.

Single bit Error Correction:

Consider that the codeword generated as before was transmitted and instead of receiving "1100110", we received "1110110". A one bit error has occurred during the reception of the codeword. Let's see how the decoding algorithm corrects this single bit error. The equation for the detecting the position of the error is given by

$$A = P_4 + D_7 + D_6 + D_5$$
$$B = P_2 + D_7 + D_6 + D_3$$
$$C = P_1 + D_7 + D_5 + D_3$$

If there is an error then "ABC" will be the binary representation of the subscript or the position of the erroneous bit.

Calculating A, B and C for the received codeword "1110110" gives A=1, B=0, C=1. Thus "ABC" is "101" in binary, which is 5 in decimal. This indicates that the fifth bit (D_5) bit is corrupted and the decoder flips the bit at this position and restores the original message.

The characteristics of the above mentioned Hamming code can be summarized as follows:

* Detection of 2 bit errors (assuming no correction is attempted);
* Correction of single bit errors;
* Cost of 3 bits added to a 4-bit message. Code rate = k/n = 4/7

******

Back to Table of Contents

2.2 Construction of Hamming codes using matrices

In the previous article we saw the fundamentals of Hamming code and how it could be constructed. The procedure seems easier since we have done it for the most basic hamming code i.e. (7, 4) code. The same procedure turns out to be laborious and tedious for increasing values of n and k.

For a (n, k) hamming code, the values n and k can be calculated using the following relationships:

For any value of (number of parity check bits) m>=3, Hamming code (n, k) can be constructed using

Block length:	$n = 2^m - 1$
Number of message bits:	$k = 2^m - m - 1$
Number of parity check bits:	$n - k = m$
Minimum distance:	$d_{min} = 3$

Hamming codes can be constructed using matrix manipulation method very easily. Two matrices are of great importance in the construction of Hamming Codes and detection of errors respectively. The first matrix called the generator matrix (G matrix) is used to generate the codewords. The second matrix called the parity check matrix (H matrix) is used to detect the errors present in the transmitted codewords.

The G and H matrix obey the following rule:

$$GH^T = 0$$

In the above equation, 'T' is the transpose operation.

Ok! How to construct the codewords from the message bits? The G matrix consists of n rows and K columns. Just take your message bits (k bit length) and simply multiply with the G matrix. This operation gives you n-bit codeword. If you have a streaming array of message bits, divide your message into blocks of k bits length and do the multiplication with the G matrix repeatedly till your input bits are exhausted.

Then how to make use of H matrix? After you receive the codewords, to check for errors, simply multiply the codewords (n bits length) with H matrix. The H matrix contains n rows and n-K columns. This multiplication gives the bit position of the bit that is corrupted. If there is no error, the output will be just '0′. This error position is called Syndrome and the method is called Syndrome Decoding.

Illustration – Construction of a simple non-systematic Hamming code:

Let us design a Hamming code scheme to generate a 7 bit codeword from a 4 bit message. As you can see here, the value of n and k are 7 and 4 respectively and we are in the process of constructing a (n, k) = (7, 4) Hamming code. The simplest way is to first construct the H matrix and then derive the G matrix from that.

H matrix is constructed with n rows and (n-k) columns. So in this case, the H matrix is a 7×3 matrix. Simply write the binary representation of numbers from 1 to 7 column wise (just omit the number '0'). So the H matrix in non-systematic form consists of the following elements represented in columns – "001″,"010″,"011″,"100″,"101″,"110″,"111″. You can write them in any order. Transposing the resulting matrix will give a H matrix of dimension n x (n-k). Accordingly the G Matrix will also change and you will get an entirely different set of codewords.

The H matrix constructed is transformed into G Matrix by the following relationships.

$$H := (A \mid I_{n-k})$$
$$G := (I_k \mid -A^T)$$

Where I_{n-k} is a (n-k) x (n-k) Identity matrix and I_k is a (k x k) Identity matrix. The above representation gives you a systematic Hamming Code.

But our H matrix (used as an illustration) is not in this form. So we need to transform our H matrix to fit the above mentioned relationship. Two types of elementary transformations can be done:

1) Swap columns with each other
2) Add two or more rows and use the result to replace another row.

In our case we just swap the columns to bring it to proper form. Remember the order of Column swapping, since we need to rearrange the elements of the G matrix in the same order to make it represent the correct generator matrix pair for our H matrix.

Example:

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$C_7 \leftrightarrow C_1$$

$$H_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$C_2 \leftrightarrow C_6$$

$$H_2 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$C_4 \leftrightarrow C_5$$

$$H_3 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$H_4 = \left[\begin{array}{cccccc|c} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$A \qquad\qquad I_3$$

$$G_3 = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

$$I_4 \qquad\qquad A^T$$

$$C_4 \leftrightarrow C_5$$

$$G_2 = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

$$C_2 \leftrightarrow C_6$$

$$G_1 = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

$$C_7 \leftrightarrow C_1$$

$$G = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right]$$

Here, G and H matrices are a pair (non-systematic coding), and G_3 and H_4 are a pair (systematic coding). Both GH and G_3H_4 pairs will obey the following rule:

$$GH^T = 0$$

If you want systematic coding, use G_3 and H_4 matrices for generation and error detection of Hamming Codes. If you want non-systematic codewords, use G and H matrices.

******

Back to Table of Contents

2.3 Introduction to Reed Solomon Codes

The Hamming codes (described in the previous section) are suitable for detecting random bit errors in a sequence of transmitted bits. If the communication medium is prone to burst errors (channel errors affecting contiguous blocks of bits), then the Hamming code may not be suitable.

For example in CD, DVD and in hard disk drives, the data is written in contiguous blocks and are retrieved in contiguous blocks. The heart of a hard disk is the read/write channel (an integral part of the disk drive controller chip). The read/write channel is used to improve the signal to noise ratio of the data that is written into and read from the disk. Its design is aimed at providing reliable data retrieval from the disk. Algorithms like PRML (Partial Response signaling with Maximum Likelihood detection) are used to increase the areal densities of the disk (packing more bits in a given area on the disk platter). Error control coding is used to improve the performance of the detection algorithm and to protect the user data from erasures. In this case, a class of error correcting codes called Reed

Solomon Codes (RS Codes) is used. RS Codes have been utilized in hard disks for the past 15 to 20 years. RS codes are useful for channels having memory like CD, DVD etc.

The other applications of RS Codes include:

1) Digital Subscriber line (DSL) and its variants like ADSL, VDSL…
2) Deep space and satellite communications
3) Barcodes
4) Digital Television
5) Microwave communication, Mobile communications and many more…

Reed Solomon Codes are linear block codes, a subset of the BCH codes called non-binary BCH. The (n, k) RS code contains k data symbols and n-k parity symbols. RS Codes are also cyclic codes since the cyclic shift of any codeword will result in another valid RS codeword.

Note the usage of the word "symbols" instead of "bits" when referring to RS Codes. The word "symbol" is used to refer a group of bits. For example if I say that I am using a (7, 3) RS Code with 5 bit symbols, it implies that each symbol is a collection of 5-bits and the RS Codeword is made up of 7 such symbols, of which 3 symbols represent data and remaining 4 symbols represent parity symbols.

An m-bit RS (n, k) Code can be defined using

$$(n, k) = (2^m - 1, 2^m - 1 - 2t)$$

In the above equation, t is the symbol error correcting capability of the RS code. This means that this code can correct t symbol errors. We can also see that the minimum distance for RS code is given by

$$d_{min} = n - k + 1 = 2t + 1$$

This gives the maximum possible d_{min}. A code with maximum d_{min} is more reliable as it will be able to correct more errors.

Example:

Consider a (255,247) RS code, where each symbol is made up of m=8 bits. This code contains 255 symbols (each 8 bits of length) in a codeword of which 247 symbols are data symbols and the remaining 8 symbols are parity symbols. This code can correct any 4 symbol burst errors.

If the errors are not occurring in a burst fashion, it will affect the codeword symbols randomly and it may corrupt more than 4 symbols. At this situation the RS code fails. So it is essential that the RS codes should be used only for burst error correction. Other techniques like interleaving and deinterleaving are used in tandem with RS codes to combat both burst and random errors.

Performance Effects of RS codes:

1) Block length Increases (n) -> BER decreases

2) Redundancy Increases (k) -> code rate decreases -> BER decreases -> complexity increases (code rate = n/k)

3) Optimum code rate for an RS code is calculated from the decoder performance (for a particular channel) at various code rates. The code rate that requires the lowest E_b/N_0 for a given desired BER

is chosen as the optimum code rate for RS Code design.

Refe [Sklar2001] for more details on mathematics behind Reed Solomon coding and decoding.

Matlab Code:

Here is a simple Matlab code for better understanding of RS code: (It can be found in Matlab Help)

```
%Matlab Code for RS coding and decoding
n=7; k=3; % Codeword and message word lengths
m=3; % Number of bits per symbol
msg = gf([5 2 3; 0 1 7;3 6 1],m) % Three k-symbol message words

% message vector is defined over a Galois field where the number must
%range from 0 to 2^m-1
codedMessage = rsenc(msg,n,k)% Encoding the message vector using RS coding
dmin=n-k+1 % display dmin
t=(dmin-1)/2 % diplay error correcting capability of the code

% Generate noise - Add 2 contiguous symbol errors with first word;
% 2 symbol errors with second word and 3 distributed symbol
% errors to last word
noise=gf([0 0 0 2 3 0 0 ;6 0 1 0 0 0 0 ;5 0 6 0 0 4 0],m)
received = noise+codedMessage

%decoded contains the decoded message and cnumerr contains the number of
%symbols errors corrected for each row. Also if cnumerr(i) = -1 it indicates that the ith row contains
unrecoverable error
[decoded,cnumerr] = rsdec(received,n,k)
```

Given below is the output of the program. Some details from the actual Matlab output are omitted for convenience. Let's interpret the output. The default primitive polynomial over which the Galois Field is defined is D^3+D+1 (which is 1011 -> 11 in decimal). This default polynomial is used to generate the message symbols. The message vector indicated by "msg" contains three message words and each word contains three symbols. It is like sending the sentence "All are fat" which contains three words with three symbols each.

$$msg = \begin{bmatrix} 5 & 2 & 3 \\ 0 & 1 & 7 \\ 3 & 6 & 1 \end{bmatrix}$$

The Reed Solomon encoded output is given below. In the encoded output, each message word is expanded by four more additional symbols.

$$encoded = \begin{bmatrix} 5 & 2 & 3 & 5 & 4 & 4 & 2 \\ 0 & 1 & 7 & 6 & 6 & 0 & 7 \\ 3 & 6 & 1 & 7 & 4 & 0 & 2 \end{bmatrix}$$

The error correcting capability of the RS code given by the parameter "t" indicates that it can correct

any two symbol errors

$$d_{min} = 5$$
$$t = 2$$

Next, the following noise vector is added to each of the symbols. From the noise matrix it can be seen that some symbols will be corrupted (matrix elements with non-zero values) and some will remain intact (matrix elements with value = 0)

$$noise = \begin{bmatrix} 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 6 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 6 & 0 & 0 & 4 & 0 \end{bmatrix}$$

The received message words which is the addition of encoded vector and noise vector is given below.

$$received = \begin{bmatrix} 5 & 2 & 3 & 7 & 7 & 4 & 2 \\ 6 & 1 & 6 & 6 & 6 & 0 & 7 \\ 6 & 6 & 7 & 7 & 4 & 4 & 2 \end{bmatrix}$$

Now, the received bits are decoded by a Reed Solomon decoder at the receiver and the redundant bits are stripped out. Finally the decoded output contains the same number of words with same symbol size. Comparing the original message vector and the decoded vector, the last row in the decoded vector deviates significantly from the last row in the original message vector.

$$decoded = \begin{bmatrix} 5 & 2 & 3 \\ 0 & 1 & 7 \\ 6 & 6 & 7 \end{bmatrix}$$

The Reed Solomon decoder catches this error (but cannot correct) and indicates the position of this error. Matlab's "rsdec" function provides this information through the output variable named -"cnumerr". This variable is again a vector whose length will be equal to the number of message words. A value of "-1" in the variable indicates the failure of proper transmission and reception.

$$cnumerr = \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix}$$

The above vector indicates error in the last row. This means that the last word of the transmitted message is irrecoverable. This error is due to the fact that we have added 3 distributed errors in the last row (check the noise vector) whereas the RS code can correct only 2 errors (as indicated by the parameter "t").

******

Back to Table of Contents

2.4 Block Interleaver Design for RS codes

Introduction:

A (n, k) Reed Solomon (RS) encoder, takes k user data symbols and converts it into n-symbol-wide codeword, by adding n-k parity symbols. The error correcting capability (t) of the RS code is computed as t<= (n-k)/2. That is, a RS code with n-k parity symbols can correct a burst error of up to (n-k)/2 symbol errors.

Block Interleaver:

An interleaver, employed at the transmitter, essentially spreads out the burst errors (by shuffling the bits across the codeword) and distributes the errors across the entire code word. Since the burst errors are now distributed across the codeword, the codeword becomes more reliable for recovery. This is because; the maximum length of a burst error in the codeword is reduced. Now the Reed Solomon code can operate more efficiently and will aid in recovering the codeword correctly. A de-interleaver used at the received (the counterpart of the interleaver) is used at the receiver (after the decoder) to reorder or restore the bits to their original position.

Assume that the dominant error mechanism in a channel is of burst type. A burst of length b is defined as a string of b unreliable consecutive symbols. If the expected burst length, b is less than or equal to t (the number of correctable symbol errors by RS coding), the code can be used as it is. However, if bursts length b >t, the error correcting code will fail. This is where interleaving comes to our rescue.

Let us assume that $b > t$ and $b \leq t \times d$, where d (the interleaving depth) is an integer. The Reed-Solomon (n, k) code can be used if we can spread the burst error sequence over several code blocks so that each block has no more than t errors (which can then be corrected). This can be accomplished using block interleaving as follows. Instead of encoding blocks of k symbols and then sending the encoded symbols consecutively, we can interleave the encoded blocks and transmit the interleaved data. In the case where n = 255, k = 235, t = 10, d = 5,the data bytes output from the Reed-Solomon encoder would appear as shown below , where bytes numbered 0 to 234 are the data bytes and bytes 235 to 254 are the parity check bytes.

Here, the data is written row by row and read back column by column. Consider now the effect of a burst error of length $b > t$, (where t is the number of correctable errors per block) and $b \leq v \times d$ for some integer "v", on the received symbols in the table. Because of the order in which the symbols are sent, a burst length less than or equal to $d \times v$ will effect at most $d + 1$ consecutive columns of the table, depending on where the burst starts. Notice that any single row (which corresponds to a

codeword) has no more than v errors in it. If d > t, these errors are within the error correction capability of the code and can be corrected. In this case, d becomes the interleaving depth. The trade-off is that extra buffer space is required to store the interleaver table and additional delay is introduced. The worst case burst length determines the size of the table (and the interleaving depth) and the table size determines the amount of buffer space required and the delay.

Design Example:

Consider a (255, 235) Reed Solomon coding system. This code can correct up to t= (n-k)/2 = 10 symbols errors. Let's assume that the channel that we are going to use, is expected to cause b=253 symbol errors. Then the required interleaver depth (d) is calculated as

$$d > \frac{b}{t} = \frac{253}{10} = 25.3$$

In this case, an interleaver depth of 26 is enough to combat the burst errors introduced by the channel. The block interleaver dimensions would be d x n = 26 x 255 (26 rows by 255 columns).

Matlab Code:

A sample Matlab code that simulates the above mentioned block interleaver design is given below. The input data is a repetitive stream of following symbols –
" **THE_QUICK_BROWN_FOX_JUMPS_OVER_THE_LAZY_DOG_** ". A (255,235) Reed Solomon decoder (with correction capability of 10 symbols) is used. We assume that the channel is expected to produce a maximum of consecutive 20 symbols of burst error. The burst errors are denoted by ' * ' .

```
%Demonstration of design of Block Interleaver for Reed Solomon Code
clc;
clear;
%_____________________________________________
%Input Parameters
%_____________________________________________
%Interleaver Design for Reed-Solomon Codes
%RS code parameters
n=255;  %RS codeword length
k=235;  %Number of data symbols
b=20;  %Number of symbols that is expected to be corrupted by the channel
%_____________________________________________
p=n-k;  %Number of parity symbols
t=p/2; %Error correction capability of RS code
fprintf('Given (%d,%d) Reed Solomon code can correct : %d symbols \n',n,k,fix(t));
fprintf('Given - expected burst error length from the channel : %d symbols \n',b);
disp('_____________________________________');
if(b>t)
    fprintf('Interleaving  MAY help in this scenario\n');
else
    fprintf('Interleaving will NOT help in this scenario\n');
end
disp('_____________________________________');
```

```
D=ceil (b/t) +1; %Interleaver Depth
memory = zeros(D,n); %constructing block interleaver memory
data='THE_QUICK_BROWN_FOX_JUMPS_OVER_THE_LAZY_DOG_';

% A constant pattern used as a data
%If n>length(data) then repeat the pattern to construct a data of length 'n'
data=char([repmat(data,[1,fix(n/length(data))]),data(1:mod(n,length(data)))]);

% Sending D blocks of similar data
intlvrInput=repmat(data(1:n),[1 D]);
fprintf('Input Data to the Interleaver -> \n');
disp(char(intlvrInput));
disp('_____');

%INTERLEAVER
%Writing into the interleaver row-by-row
for index=1:D
    memory(index,1:end)=intlvrInput((index-1)*n+1:index*n);
end
intlvrOutput=zeros(1,D*n);

%Reading from the interleaver column-by-column
for index=1:n
    intlvrOutput((index-1)*D+1:index*D)=memory(:,index);
end

% Create b symbols error at 25th Symbol location for test in the interleaved output
%'*' means error in this case
intlvrOutput(1,25:24+b)=zeros(1,b)+42;
fprintf('\nInterleaver Output after being corrupted by %d symbol burst error - marked by "*"->\n',b);
disp(char(intlvrOutput));
disp('_____');

%Deinteleaver
deintlvrOutput=zeros(1,D*n);
%Writing into the deinterleaver column-by-column
for index=1:n
    memory(:,index)=intlvrOutput((index-1)*D+1:index*D)';
end

%Reading from the deinterleaver row-by-row
for index=1:D
    deintlvrOnput((index-1)*n+1:index*n)=memory(index,1:end);
end
```

```
fprintf('Deinterleaver Output->\n');
disp(char(deintlvrOnput));
disp('_____');
```

Simulation Results:

As we can see from the above simulation that, even though the channel introduces 20 symbols of consecutive burst error (which is beyond the correction capability of the RS decoder), the interleaver/de-interleaver operation has effectively distributed the errors and reduced the maximum burst length to 7 symbols (which is easier to correct by (255,235) Reed Solomon code.

******

Back to Table of Contents

2.5 Convolutional Coding and Viterbi Decoding

Convolutional codes differ from block codes in terms of method of operation. A convolutional encoder operates over serial data, whereas block codes operates over a block of input data. Also different is the utilization of memory elements in the convolutional encoders. In the case of block codes, there is no memory element involved in the generation of encoded data.

Convolutional codes are specified as (n, k, L), where n is the number of output bits from the encoder, k is the number of input bits to the encoder and L is the constraint length of the encoder. Disparate expressions for constraint length are often found in different text books but the fundamental idea is the same. The constraint length is used to calculate the number of memory stages or flip-flops used in the encoder. As long as we know L and the underlying formula, we can calculate the number of memory elements (m). So it does not really matter which expression for L is used. The constraint length is expressed as

$$L = k(m + 1) \qquad (1)$$

In some text books it is expressed as k x (m) and in some other books it is even expressed as L=m+1.We will use the first expression throughout our discussion.

We will take up a simple convolutional code (2, 1, 3) where n=2, k=1 and L=3 (the expression L=k (m+1) is used).Let us construct the encoder from the above information. The encoder is constructed with 1 input bit, 2 output bits and 2 memory elements. Note that the L=k (m+1) expression leads to 2 memory elements. Here these two memory elements are used to store the past 2 input bits. If the expression L=k*m was used and for a (2, 1, 3) encoder (L=3), the number of memory elements would be 3, where these 3 memory elements are used to store past 3 input bits. So the expression for constraint length has to be carefully interpreted, otherwise any misinterpretation will lead to a different encoder structure altogether.

Now we know the number of bits going into the encoder, number of bits coming out from it and the number of memory elements. Till now the encoder is like a black box to us in the sense that we don't know how the memory elements are utilized to generate the output bits from the input. To fully understand the encoder structure we need something called "generator polynomials" that tell us how the memory elements are linked to achieve encoding. The generator polynomials for a specific convolutional encoder set (n, k, L) are usually found through simulation. The set (n, k, L) along with n generator polynomials completely describes a convolutional encoder.

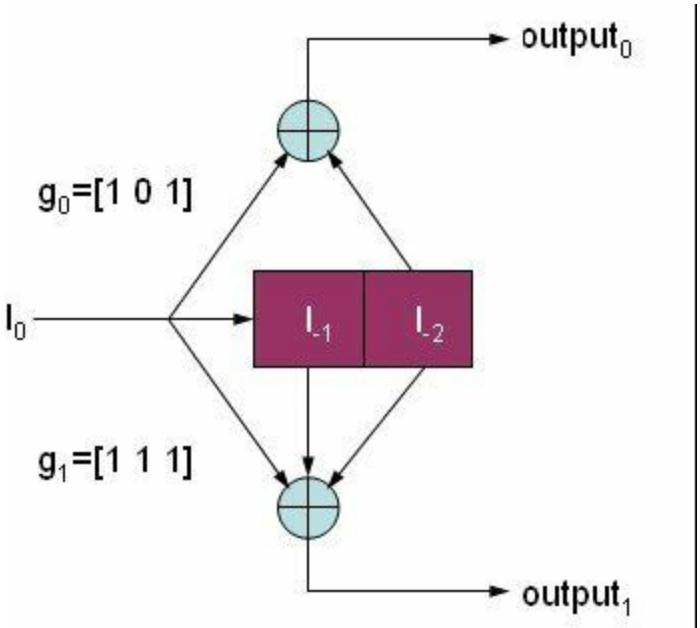
For our (2, 1, 3) encoder, we use the following two generator polynomials, one for generating each output bit.

$$g_0 = [1\ 0\ 1]$$
$$g_1 = [1\ 1\ 1]$$

Let us put everything to make the encoder structure. The first bit in the generator polynomial indicates present input I_0, the second bit indicates the past input I_{-1} (first memory element), and third bit indicates the past input I_{-2} (second memory element) and so on.

The polynomial g_0=[1 0 1] indicates that the output is formed by XORing present input I_0 and the past input I_{-2}. The polynomial g_1=[1 1 1] indicates that the output is formed by XORing present input I_0 and

the two past inputs I_{-1}, I_{-2}. Thus the encoder structure is given by



Encoder structure for (2,1,3) Convolutional Code with g_0=[1 0 1] and g_1=[1 1 1]

For proper generation of codewords (and also for proper decoding at the receiver), the encoder has to start with a known state and end in a known state. The encoder is usually initialized with an all zero state. Similarly, the last bit of input data has to be flushed out of the encoder memory properly. If not, it will generate a weaker codeword that may not protect the last few bits. The flushing is done by appending the incoming data stream with "m" zeros.

Let the input stream of bits be given by 010111 (with MSB on the right). Adding m=2 trailing zeros (for flushing), the encoded output is given by 00 11 01 00 10 01 10 11. The output is paired in groups of two. The next table illustrates the encoding process.

I_0	I_{-1}	I_{-2}	$Output_0$	$Output_1$
0	0	0	Initial State	
0	0	0	0	0
1	0	0	1	1
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1
0	1	1	1	0
0	0	1	1	1
0	0	0	Memories Flushed	

Encoder Structure, State Diagram and Trellis:

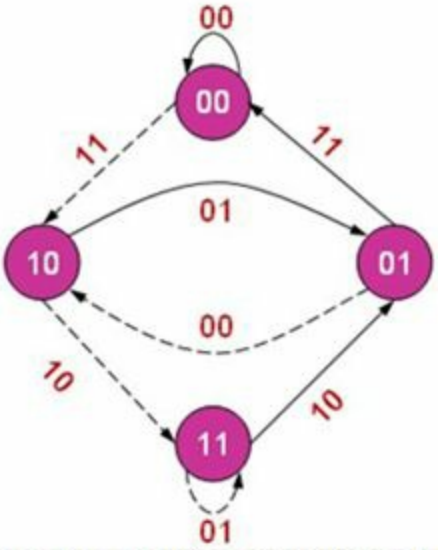
The convolutional encoder can also be represented using a finite state machine. The entire behavior of a convolutional encoder is depicted by a state diagram. The number of states in a state diagram depends on the number of memory elements in the encoder. If number of memory elements is m then the number of states in the state diagram will be 2^m.For the (2,1,3) convolutional encoder the number of states will be 4 , i.e., the last two memory elements are used to store the past inputs I_{-1} and I_{-2}.

Transition between the states depends on the present input I_0.The solid line in the state diagram indicates the transitions due to the input I_0=0 and dotted lines are used to represent the transitions due to the input I_0= 1. The output bits generated during each state transition is written in dark red color (along the transition lines).

State diagrams can be easily constructed with the help of a state table as shown below.

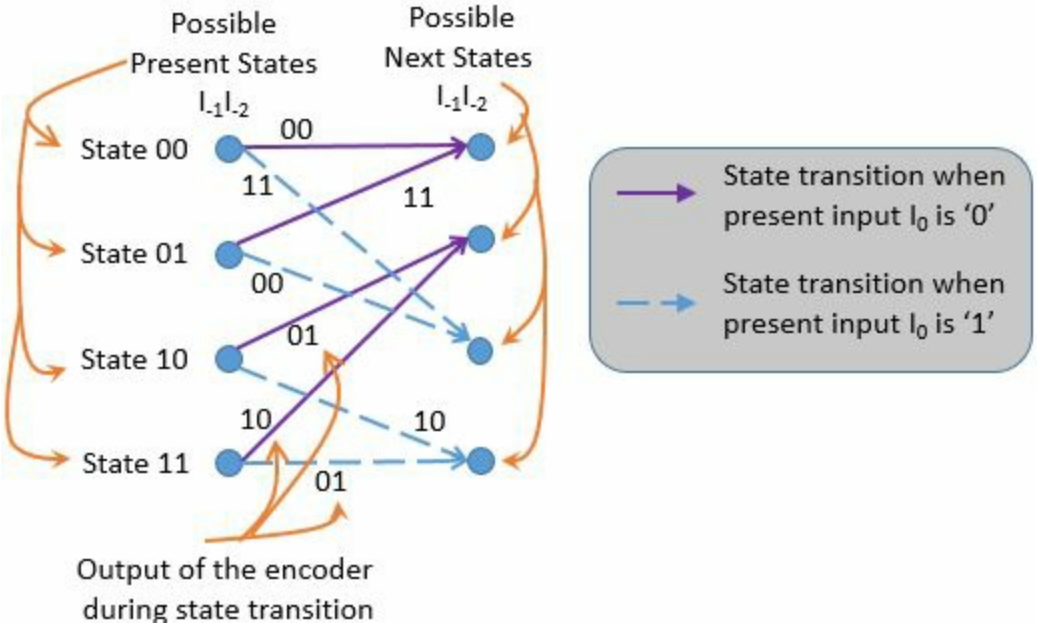
State table – (2,1,3) Convolutional Encoder

Input	Current State	Next State	Output
0	00	00	00
0	01	00	11
0	10	01	01
0	11	01	10
1	00	10	11
1	01	10	00
1	10	11	10
1	11	11	01



State Diagram for (2,1,3) Convolutional Encoder

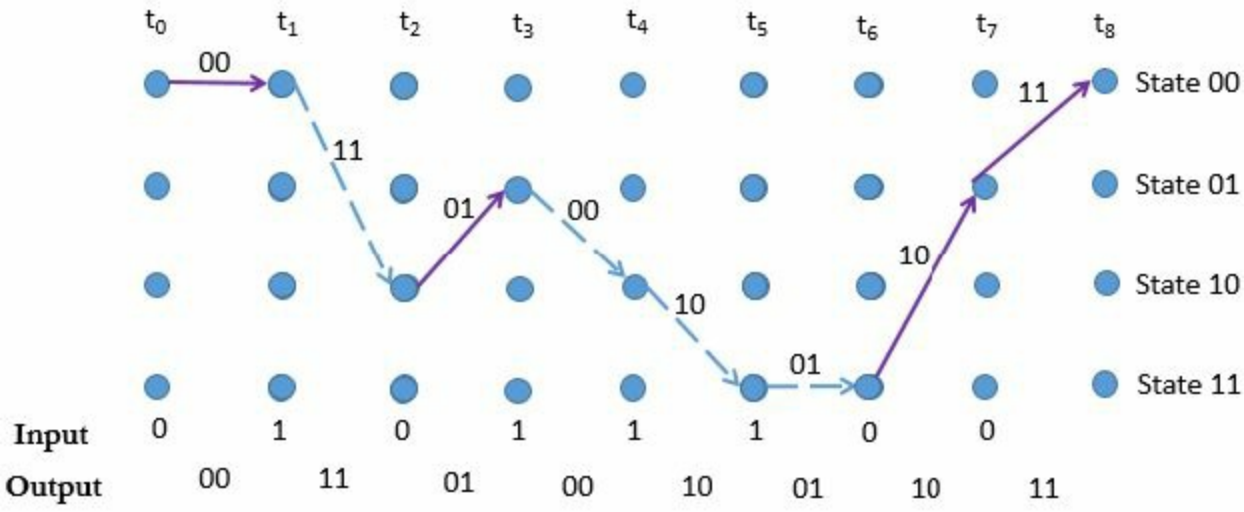
A trellis diagram is constructed from the given state diagram. Trellis diagram helps us to understand the concept of Viterbi Decoding algorithm that is used to decode the convolutional encoded data.



A single stage Trellis is illustrated for understanding. The four possible states of the encoder are depicted as four horizontal rows. Present states are considered to be on the left side and the possible next states on the right side of the trellis. The connections between the present states and the next states exactly follow the state diagram representation. Again, the output bits generated during each state transition are indicated along the transition lines. A multiple stage Trellis is constructed by replicating the single stage and juxtaposing each of the single stage side by side. As the input bits are continuously fed to the decoding system, the trellis progresses from one stage to another stage (at

progressively increasing time instants).

Let us run the trellis with the inputs along with the two zeros for memory flushing – 01011100.The trellis provides the following output -00 11 01 00 10 01 10 11, which matches the previous example.



Viterbi Decoding of Convolutional Codes:

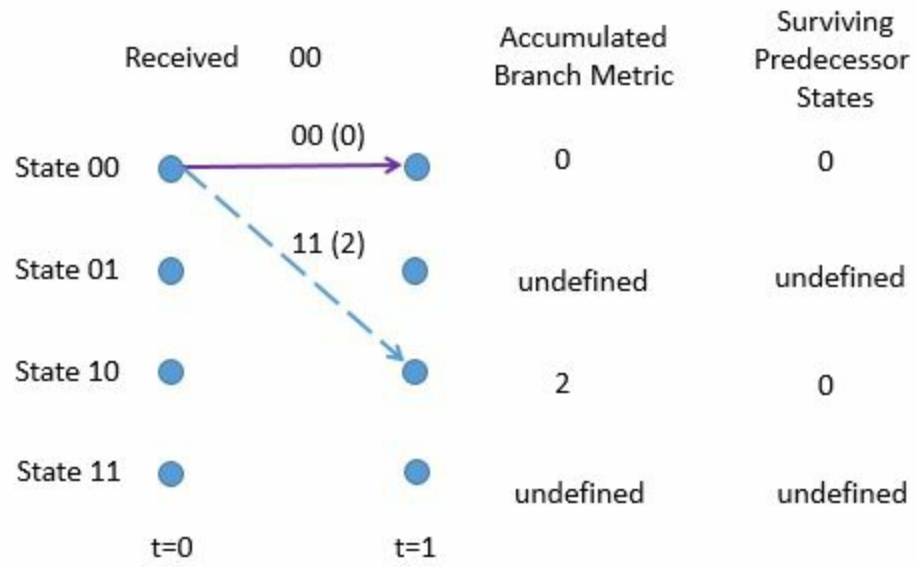
Viterbi algorithm is utilized to decode the convolutional codes. Again, the decoding can be done in two approaches: hard decision decoding and soft decision decoding. In this section we will use hard decision decoding to decode a set of received symbols. A more detailed implementation for soft decision Viterbi decoding is available at [WebVladimir]

The following treatment of the subject is based on the work done by Andreas Gerstlauer, for more details refer - [WebAndreas].

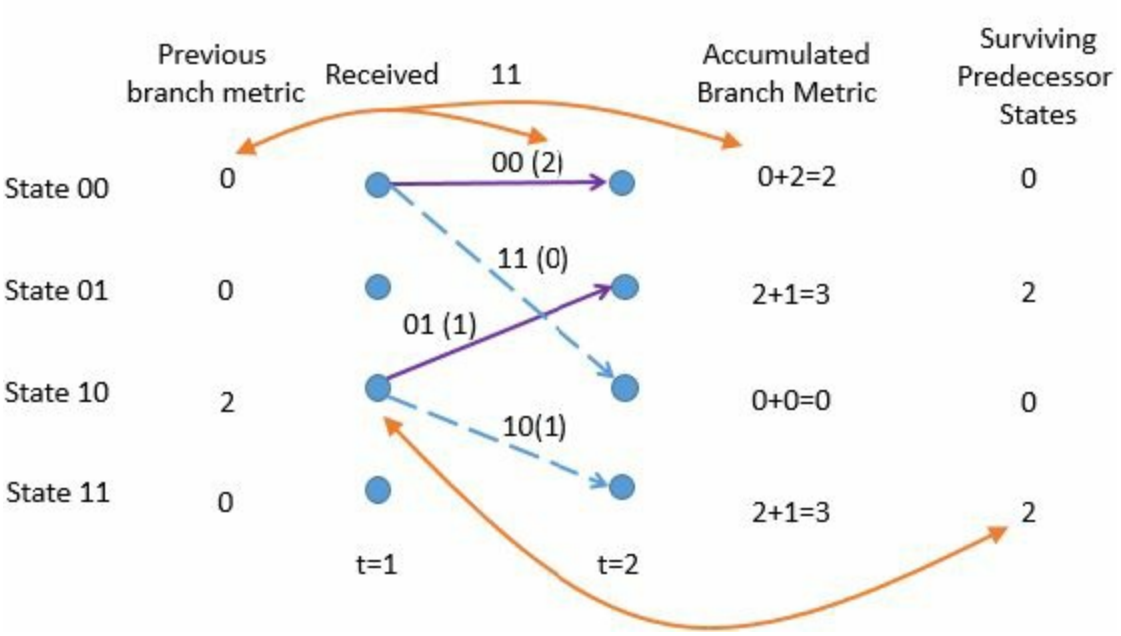
Let us say the received symbols are 00 11 11 00 10 01 10 11 – with one symbol in error (3rd. symbol). Given the received symbols, we would like to guess the input bit to the encoder at each state of the trellis. This is done using Viterbi algorithm (refer next figure).

To decode properly, the decoder has to start in a known state – state 00 at time t=0. Given that the present state of the memory is '00', there are two possible next states – state '00' and state '10' for the inputs '0' and '1' respectively. For these two state transitions, the decoder will output either '00' or '11' (see the state table or trellis diagram). Thus, if the encoder is input with '0', the output during the transition will be '00'. Similarly, the output of the encoder will be '11' for the input ='1'. At this point of the other possible next states '01' and '11' are undefined.

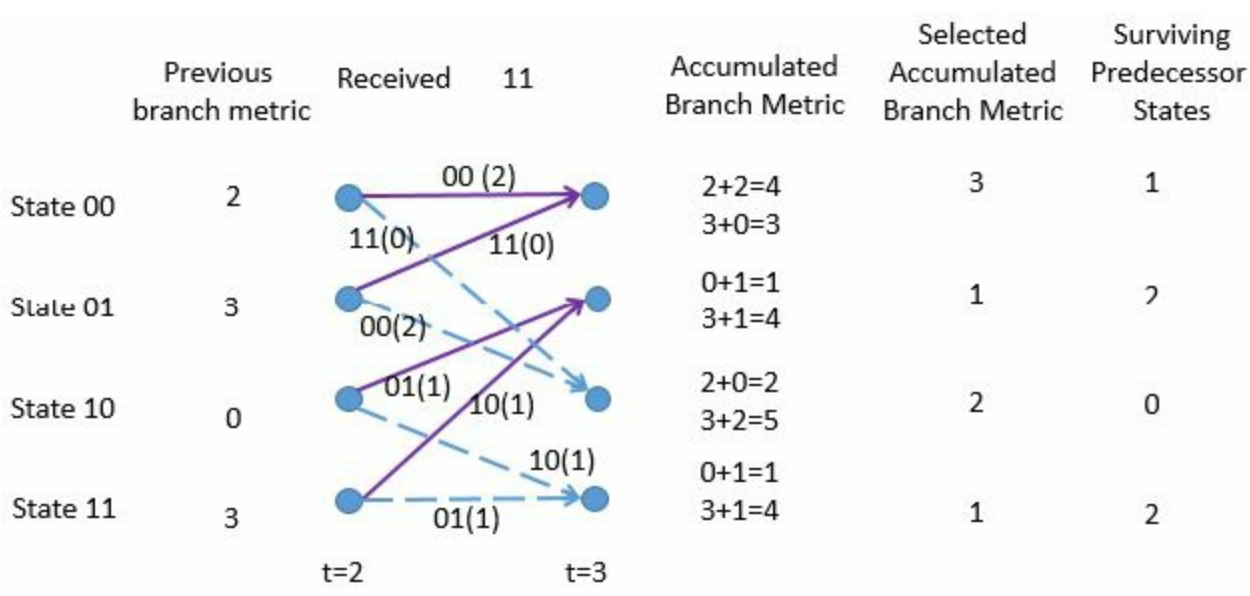
'00' is the first symbol received. Since there are two possible outputs at this point of time, we compute the Hamming distance between the symbol received and the two possible outputs. The Hamming distance between the received symbol '00' and the transition output '00' is '0'. Similarly the Hamming distance between the received symbol '0'' and the transition output '11' is '2'. In the figure, the distance metric is shown in brackets. The next possible states are shown under the heading –"Surviving predecessor states".



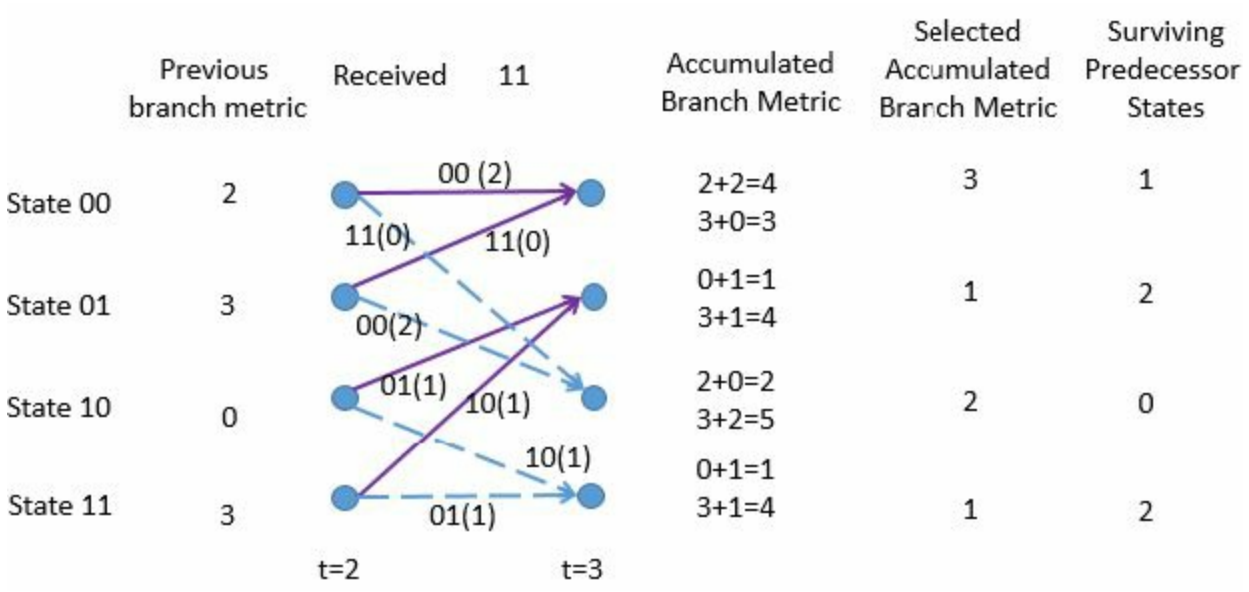
Now, at t=1, we have two states to start with for the next received symbol= '11'. The two possible present states are state '00' and state '10'. From here, depending on the input '0' or '1', the next state can be any of the four possible states (as shown in the figure). The branch metrics are computed in the same way above for all the four possible transition outputs. The branch metrics are added with that of the previous branch metrics (the undefined branch metrics are assumed to be zero). The surviving predecessor states are noted down in a separate table.



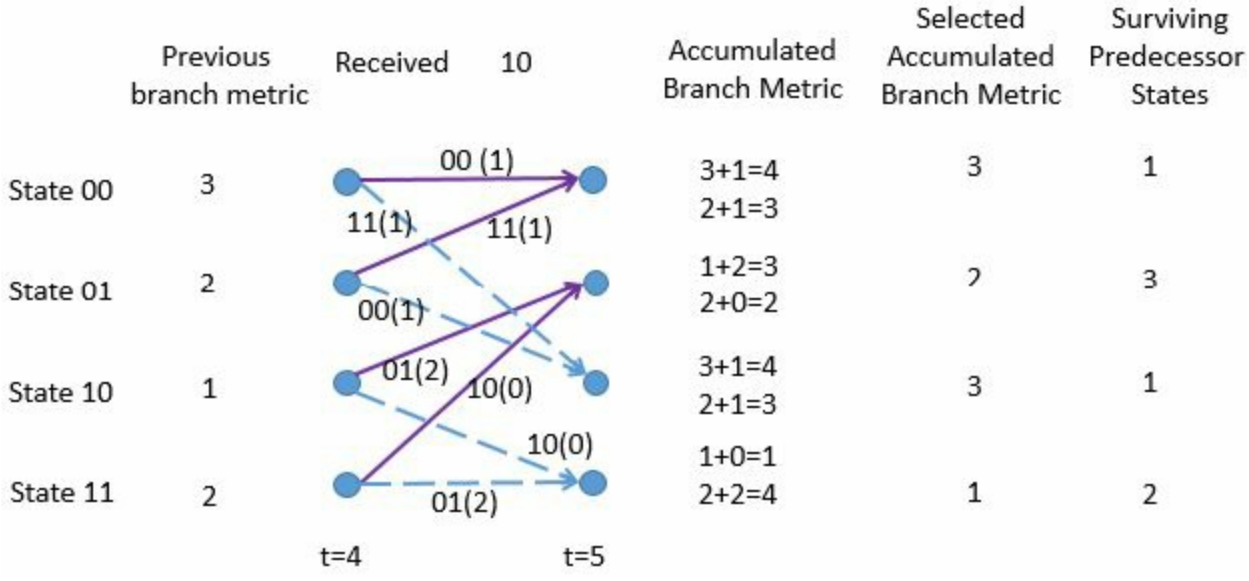
At t=2, things get a little complicated. The next received symbol is '11'. Now the trellis has four possible states to begin with. All the four possible states may take any of the four possible next states depending on the input '0' or '1' at the encoder. Now we have 8 branch metrics to calculate. But for each next state (at t=4), there are two branches that lead to the same state. Thus we need to select only one of those branches that lead to the state. This is done by selecting the branch that has a minimum accumulated branch metric. If both the accumulated metrics are same, then we have two choices to make: 1) randomly select either of the branches or 2) consistently select either the lower branch or the upper branch. This selection procedure is accomplished by "Add-Compare-and-Select" (ACS) unit in the hardware.

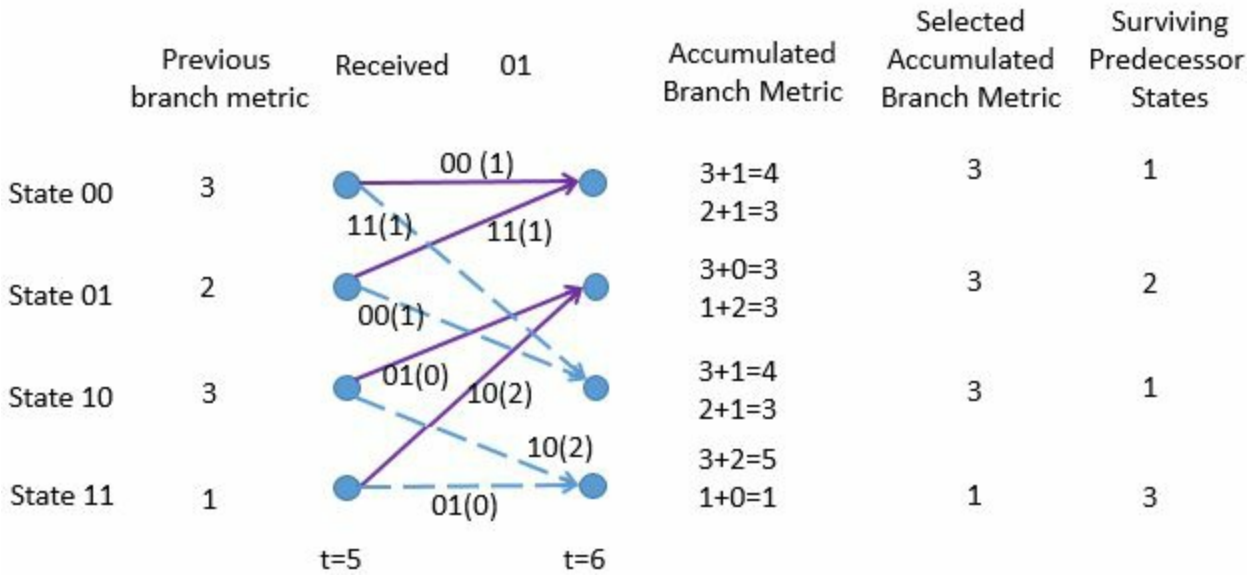


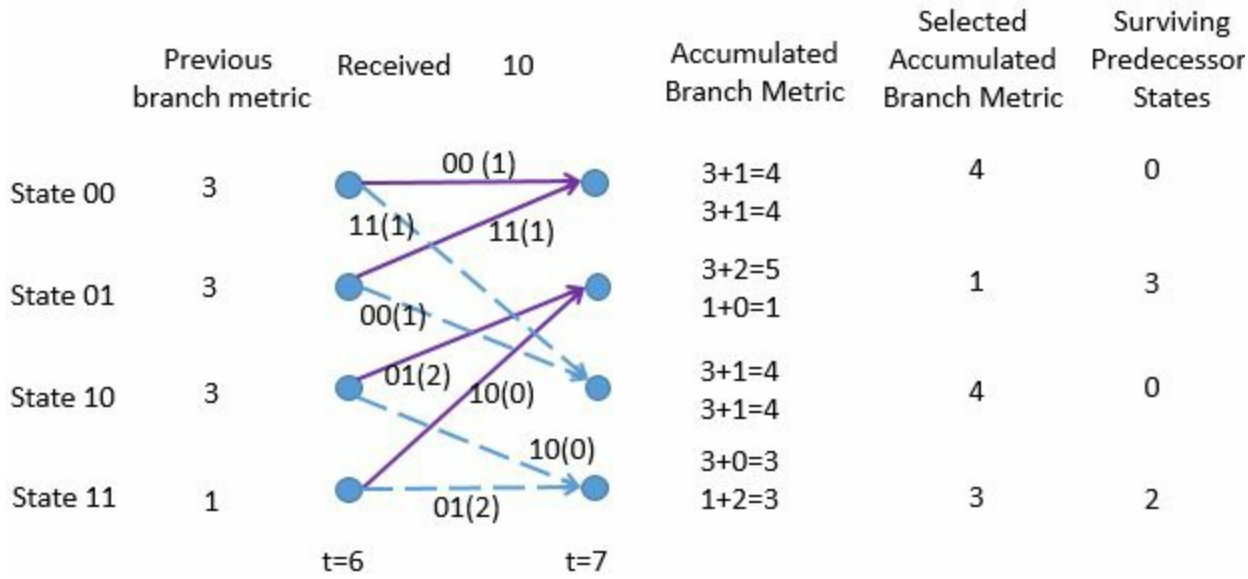
At t=3, the accumulated branch metrics are computed in the same way above. Notice that the accumulated branch metrics for the state '00' at t=4 are the same for both paths. Here the upper path (corresponding to input bit= '0') is chosen.

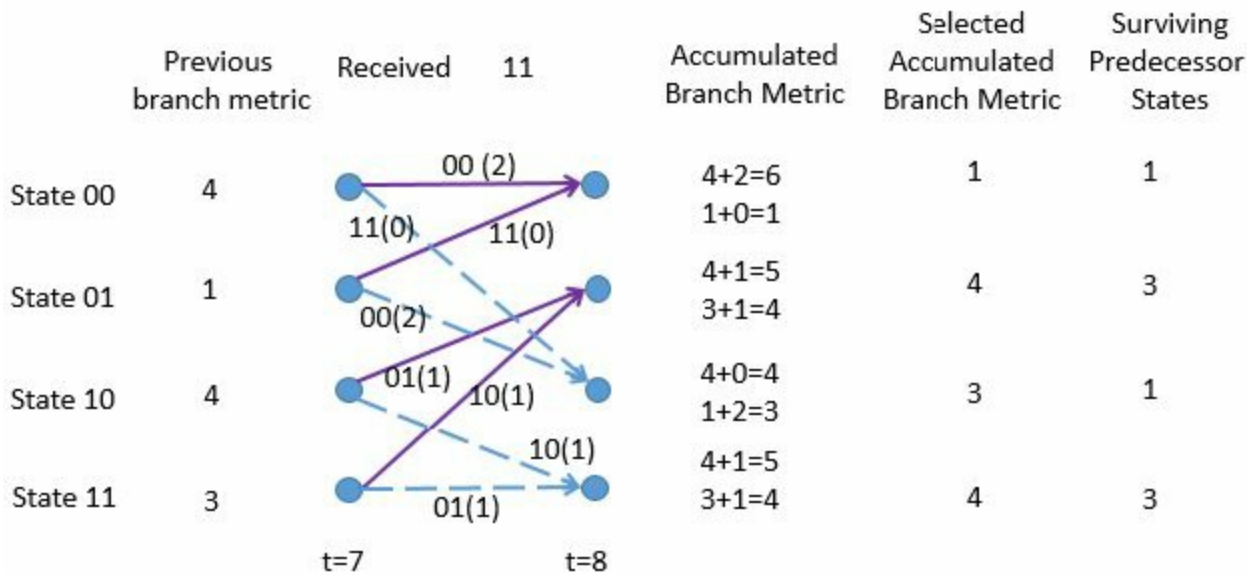


The subsequent calculation of branch metrics and surviving predecessor states for the time instants t=3,4,5,6,7 are shown in the following figures.









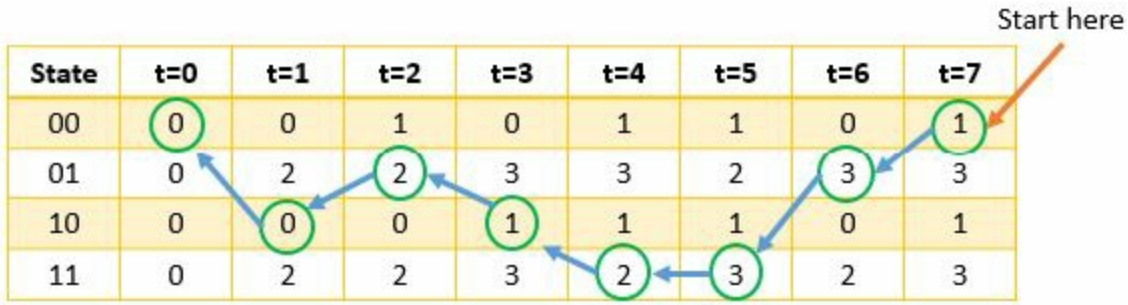
The following table is constructed from the "selected accumulated branch metric" values as we progress from one time instant to another. It is used to select the initial state of decoding. The initial state of decoding is chosen by choosing the state (at the end of the time instants) for which the accumulated branch metric is minimum.

State	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
00		0	2	3	3	3	3	4	1
01			3	1	2	2	3	1	4
10		2	0	2	1	3	3	4	3
11			3	1	2	1	1	3	4

From the above figure, at the end of the trellis (t=8), the minimum accumulated branch metric is '1' and it occurs for state '00'. So this becomes the starting decoder state to perform the trace back operation. The trace back operation is performed by using the next table that is constructed using the "surviving predecessor states" at each time instant.

State	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7
00	0	0	1	0	1	1	0	1
01	0	2	2	3	3	2	3	3
10	0	0	0	1	1	1	0	1
11	0	2	2	3	2	3	2	3

We have determined from the previous table that the starting state of decoder for the trace back operation is state '00'. Now we begin with state '00' at time instant t=8 and start the trace back operation as shown in the next table.



State	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7
00	0	0	1	0	1	1	0	1
01	0	2	2	3	3	2	3	3
10	0	0	0	1	1	1	0	1
11	0	2	2	3	2	3	2	3

Now record the states as per the trace back operation. Note that the last state and the first state are the same. This means that the trellis should start and end in the same known state.

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
Traced back states	0	0	2	1	2	3	3	1	0

Re-arranging the state transition table (shown for the encoding operation) gives the following table. The original message can be recreated by using the table given above and the following state transition table. The invalid transitions are marked with 'x'.

	Given the Next State, the input to the encoder will be			
Current State	**00 (0)**	**01 (1)**	**10 (2)**	**11 (3)**
00 (0)	0	x	1	X
01 (1)	0	x	1	x
10 (2)	x	0	x	1
11 (3)	x	0	x	1

Combining the trace-back operation and the state transition table above, the original recreated (decoded) message will be

	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7
Decoded Message	0	1	0	1	1	1	0	0

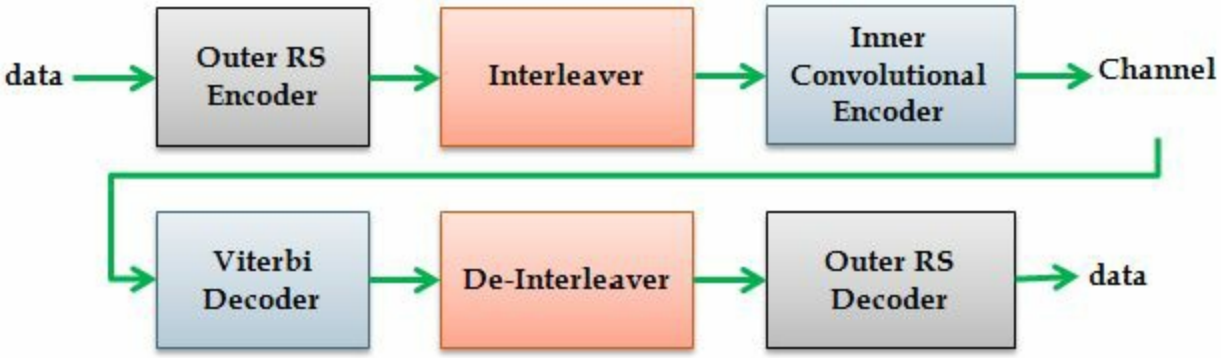
Thus you can see that the decoder has recovered from the one symbol error and has successfully recreated the input-message-to-the-encoder.

Two Level Coding Systems:

Convolution codes with Viterbi decoding are not good at burst error correction, but they are good at random error correction. On the contrary, the Reed Solomon coding is good at burst error correction and not so good at random error correction. So the advantages of these two codes are exploited in many systems to provide good error correction capabilities against both random and burst errors.

In such systems, data is encoded with a Reed-Solomon code first and then they are processed by an interleaver (which places symbols from the same Reed-Solomon codeword far from each other), and then encoded with a convolutional code.

At the receiver, data is processed by a Viterbi decoder first. The bursts of errors from its output are then de-interleaved (with erroneous symbols from one burst getting into different Reed-Solomon codewords) and decoded with a Reed-Solomon decoder. This type of coding system is called two level coding. It is often found in hard disk drive read channel chips, which paves way for higher areal density and therefore more capacity.



******

Back to Table of Contents

Chapter 3: Inter Symbol Interference and Filtering

3.1 Introduction to controlled ISI (Inter Symbol Interference)

Most of the communication channels (data storage devices, optical, wireless channels etc...) can be considered as band-limited linear filters. The channels can be modeled as having the following frequency response (over a given bandwidth W).

$$H(f) = A(f)e^{j\theta(f)}$$

Here A(f) is amplitude response and θ(f) is the phase response of the channel over the given bandwidth W. The envelope or group delay for the given filter model is defined as,

$$\tau(f) = -\frac{1}{2\pi}\frac{d\theta(f)}{df}$$

A channel is considered non-distorting (within the given bandwidth W occupied by the transmitted signal), when the amplitude response is constant and the phase response is a linear function of frequency (within the given bandwidth W). That is, the group delay, given above, is constant for a non-distorting channel.

Amplitude distortion occurs when the amplitude response is no longer a constant. Delay/phase distortion occurs when the phase response is not a linear function of frequency (that is, the envelope or group delay is not a constant).

A non-ideal channel frequency response is caused by amplitude and phase distortion. When a succession of pulses transferred through a non-ideal channel, at a rate 2W symbols/second (R=1/T=2W -'Nyquist Rate'), will get distorted to a point that they are no longer distinguishable from each other. This is called Inter Symbol Interference (ISI). It means that a symbol transmitted across a non-ideal channel will be affected by the other symbols. The symbols smear on to each other and therefore are indistinguishable at the receiver. To minimize the effect of ISI, pulse shaping filters are generally employed at the transmitter and receiver to match the spectral characteristics of the signal with that of the channel.

Generally following strategies are employed to mitigate ISI

1) Use ideal rectangular pulse shaping filters to achieve zero ISI:

Maximum transferable data rate that is possible with zero ISI is R=1/T=2W symbols/second, provided the "ideal rectangular transmit and receive pulse shaping filters" are used. Ideal rectangular transmit and receive filters are practically unrealizable. So this option is not viable to achieve zero ISI.

2) Relax the condition of transmitting at maximum rate R=1/T=2W, to achieve zero ISI:

If the transmission rate is reduced below 2W (i.e. R=1/T<2W), then it is possible to implement practically realizable filters. Raised Cosine and Square Root Raised cosine filters are generally used to achieve zero ISI, if the transmission data rate is reduced below 2W. The signals generated using this method are called full response signals.

3) Relax the condition of zero ISI and transfer at Nyquist Rate (R=1/T=2W):

In this method, we relax the condition of achieving zero ISI, so that the data can be transferred at maximum possible rate (R=1/T=2W). Instead of achieving zero ISI, this method introduces controlled amount of ISI in the transmitted signal and counters it upon receiving the signal at the receiver. The transmit filter is designed to introduce 'deterministic' or 'controlled' amount of ISI and is counteracted in the receiver side. Methods like duobinary signaling, modified duobinary signaling are employed under this category. The resulting signals are called partial response signals which are transmitted at Nyquist rate of 2W symbols/second. This method is also called "Correlative Coding".

******

Back to Table of Contents

3.2 Correlative coding – Duobinary Signaling

The condition for zero ISI (Inter Symbol Interference) is

$$p(nT) = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases}$$

It states that when sampling a particular symbol (at time instant nT =0), the effect of all other symbols on the current sampled symbol is zero.

As discussed in the previous section, one of the practical ways to mitigate ISI is to use partial response signaling technique (otherwise called as "correlative coding"). In partial response signaling, the requirement of zero ISI condition is relaxed - as a controlled amount of ISI is introduced in the transmitted signal and is counteracted in the receiver side.

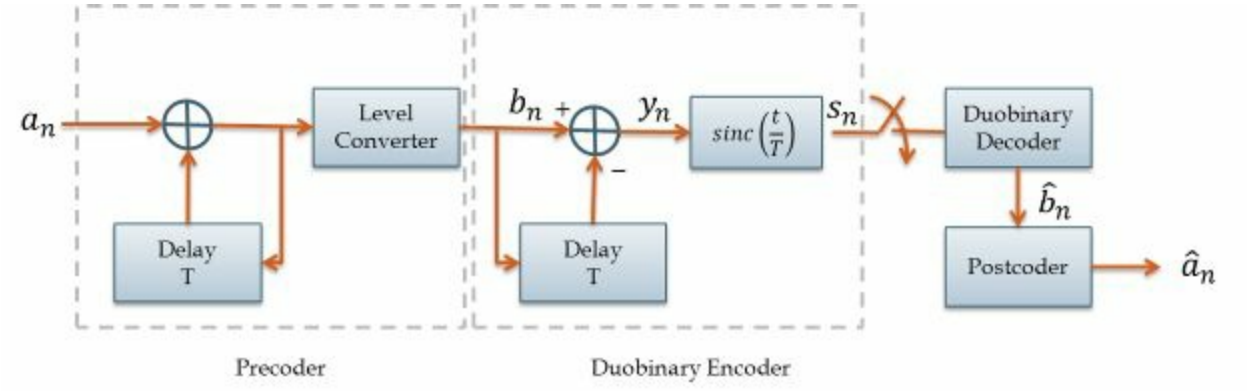
By relaxing the zero ISI condition, the above equation can be modified as,

$$p(nT) = \begin{cases} 1, & n = 0,1 \\ 0, & \text{otherwise} \end{cases}$$

This states that the ISI is limited to two adjacent samples. Here we introduce a controlled or "deterministic" amount of ISI and hence its effect can be removed upon signal detection at the receiver.

Duobinary Signaling:

The following figure shows the duobinary signaling scheme.



Encoding Process:

1) a_n = binary input bit; $a_n \in \{0,1\}$.

2) b_n = NRZ polar output of Level converter in the precoder and is given by

$$b_n = \begin{cases} -d, & if\ a_n = 0 \\ +d, & if\ a_n = 1 \end{cases}$$

3) y_n can be represented as

$$y_n - b_n + b_{n-1} = \begin{cases} +2d, & if\ a_n = a_{n-1} = 1 \\ 0, & if\ a_n \neq a_{n-1} \\ -2d, & if\ a_n = a_{n-1} = 0 \end{cases}$$

Note that the samples b_n are uncorrelated (i.e either +d for "1" or -d for "0" input). On the other-hand, the samples y_n are correlated (i.e. there are three possible values +2d, 0, -2d depending on a_n and a_{n-1}). It means that duobinary encoding correlates the present sample a_n and the previous input sample a_{n-1}.

4) From the diagram, impulse response of the duobinary encoder is computed as

$$h(t) = sinc\left(\frac{t}{T}\right) + sinc\left(\frac{t-T}{T}\right)$$

Decoding Process:

5) The receiver consists of a duobinary decoder and a postcoder (inverse of precoder).The duobinary decoder implements the following equation (which can be deduced from the equation given under step 3 (see above))

$$\hat{b}_n = y_n - \hat{b}_{n-1}$$

This equation indicates that the decoding process is prone to error propagation as the estimate of present sample relies on the estimate of the previous sample. This error propagation is avoided by using a precoder before duobinary encoder at the transmitter and a postcoder after the duobinary decoder. The precoder ties the present sample and the previous sample (correlates these two samples). The postcoder does the reverse process.

6) The entire process of duobinary decoding and postcoding can be combined together as one algorithm. The following decision rule is used for detecting the original duobinary signal samples $\{a_n\}$ from $\{y_n\}$

$$\begin{aligned} &\text{if } y_n < d, \quad \text{then } \hat{a}_n = 1 \\ &\text{if } y_n > d, \quad \text{then } \hat{a}_n - 0 \\ &\text{if } y_n = 0, \text{ randomly guess } \hat{a}_n \end{aligned}$$

Matlab Code:

File 1: duoBinaryEncoder.m

```
function [response]=duoBinaryEncoder(os_factor)
t=-4:1/os_factor:4; %Limiting the response to -4T to 4T
p=zeros(1,length(t));
for i=1:1:length(t)
    if t(i)==0 p(i)= sin(pi*(t(i)-1))./(pi*(t(i)-1))+1;
    else if t(i)==1 p(i)=sin(pi*(t(i)))./(pi*t(i))+1;
    else p(i) = sin(pi*t(i))./(pi*t(i))+sin(pi*(t(i)-1))./(pi*(t(i)-1)); end
end
end
response=p;

figure;
plotHandle=stem(t,response);
set(plotHandle,'LineWidth',1.5);
hold on; plotHandle=plot(t,response,'r');
set(plotHandle,'LineWidth',1.5);
title('Impulse Response of duobinary encoder');
xlabel('Sampling Time');
ylabel('Amplitude');
end
```

File 2: precoder.m

```
function bn=precoder(an)
%an=binary input bits to precoder
bn=zeros(1,length(an));
for i=1:length(an);
    if i==1 bn(i)=xor(an(i),1);
    else bn(i)=xor(an(i),bn(i-1));
    end
end
bn=[1 bn];
end
```

File 3: duoBinaryDecoder.m

```
function an = duoBinaryDecoder(yk)
%Decoding duobinary signal
an=zeros(1,length(yk));
for i=1:length(yk)
    if abs(yk(i))<1 %(d=1) an(i)=1;
```

```
    else if abs(yk(i))>1 an(i)=0;
    else an(i)=randn(1,1)<=0.5;
    end
end
end
```

File 4: testDuoBinary.m

```
%Simulation of Duobinary Signaling

clc;clear;close all;
overSampling_Factor=4;
an= [0 0 1 0 1 1 0 1 1 1]; %Input binary bits
ak=precoder(an); %Precoder
bn=2*ak-1; % Level converter

%Duobinary Encoder
Input_bit_os=upsample(bn,overSampling_Factor); %oversampling
pt = duobinaryEncoder(overSampling_Factor); % impulse response duobinary encoder
output_of_duoB_filter = conv(Input_bit_os,pt); %convolving bn with duobinary response
L=length(output_of_duoB_filter);
t=-4:1/overSampling_Factor:(-4+(L-1)*1/overSampling_Factor); %defining the response from -4T to
4T

figure;
subplot(2,1,1);
plotHandle=stem(t,output_of_duoB_filter);
set(plotHandle,'LineWidth',1.5);
title('Response of duoB Filter at Tx side')
xlabel('Samples')
ylabel('Amplitude')

%Simplified algorithm to decoder duobinary signaling
yn=output_of_duoB_filter;
midSample=length(-4:1/overSampling_Factor:4)/2;
yn_truncated=yn(midSample-1+overSampling_Factor:end); %Remove unwanted portions(first few
samples till the peak value)

%Now the first sample contains the peak value of the response. From here the samples are extracted
depending on the oversampling factor
yk = downsample(yn_truncated,overSampling_Factor,1); %here offset=1 means starting from 1st
sample
yk=yk(1:length(an));% length output samples  must be equal to length of an. Discard the remaining
%The above mentioned process is equivalent to sampling from the time instant nT=T (i.e. n=1)

subplot(2,1,2);
plotHandle=stem(yk);
```

```
set(plotHandle,'LineWidth',1.5);
title('Downsampled output (ADC conversion and Sampling)')
xlabel('Samples')
ylabel('Amplitude');
anEstimated = duoBinaryDecoder(yk);
disp('Input Symbols to the system : ');disp(an);
disp('Decoded Symbols :');disp(anEstimated);
```

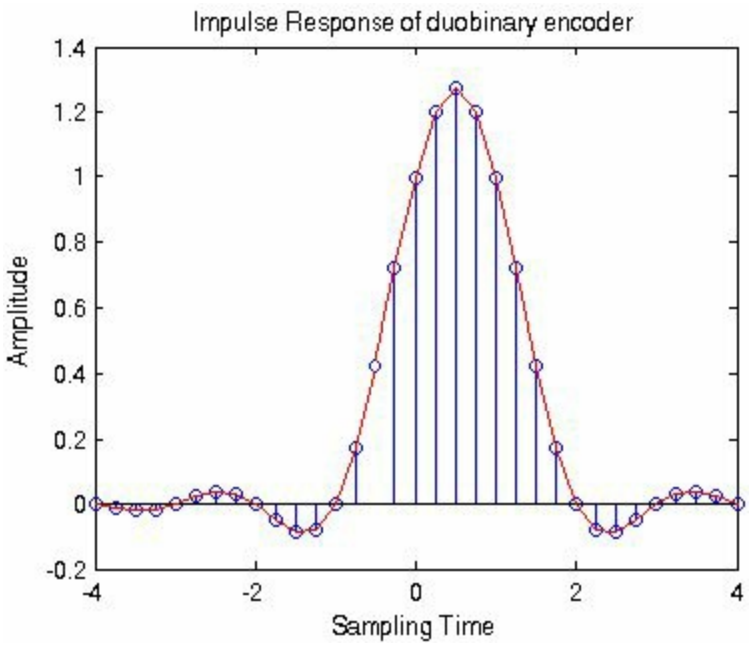
Simulation Results

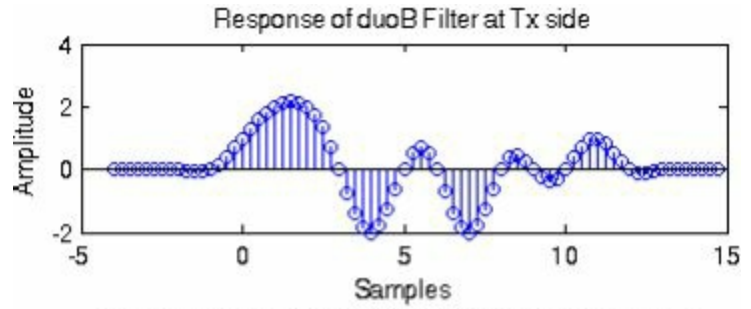
Input Symbols to the system: (a_n)

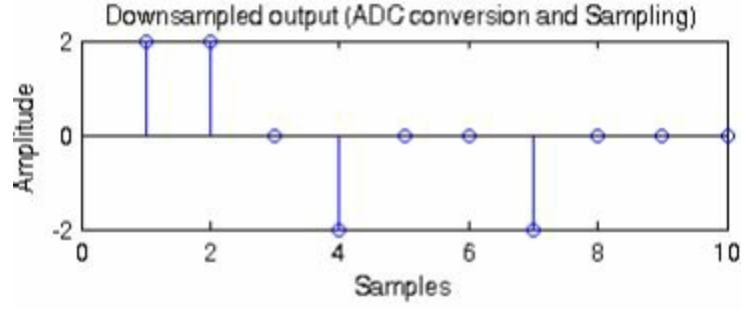
0 0 1 0 1 1 0 1 1 1

Decoded Symbols: (estimated a_n)

0 0 1 0 1 1 0 1 1 1







******

Back to Table of Contents

3.3 Modified Duobinary Signaling

Modified Duobinary Signaling is an extension of duobinary signaling. Modified Duobinary signaling has the advantage of zero PSD at low frequencies (especially at DC) which is suitable for channels with poor DC response. It correlates two symbols that are 2T time instants apart, whereas in duobinary signaling, symbols that are 1T apart are correlated.

The general condition to achieve zero ISI is given by

$$p(nT) = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases}$$

As discussed in a previous article, in correlative coding, the requirement of zero ISI condition is relaxed as a controlled amount of ISI is introduced in the transmitted signal and is counteracted in the receiver side

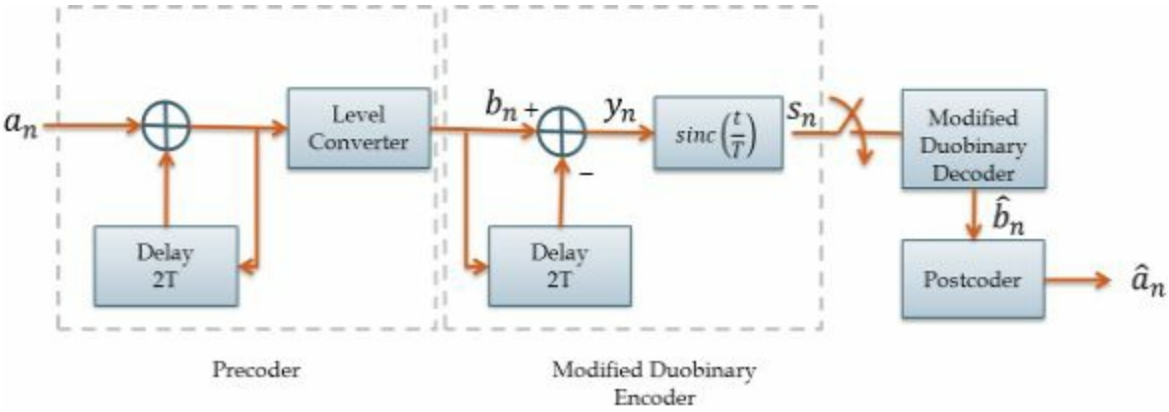
In the case of modified duobinary signaling, the above equation is modified as

$$p(nT) = \begin{cases} 1, & n = 0,2 \\ 0, & \text{otherwise} \end{cases}$$

This states that the ISI is limited to two alternate samples. Here a controlled or "deterministic" amount of ISI is introduced and hence its effect can be removed upon signal detection at the receiver.

Modified Duobinary Signaling:

The following figure shows the modified duobinary signaling scheme.



Encoding Process:

1) a_n = binary input bit; $a_n \in \{0,1\}$.

2) b_n = NRZ polar output of Level converter in the precoder and is given by

$$b_n = \begin{cases} -d, & if\ a_n = 0 \\ +d, & if\ a_n = 1 \end{cases}$$

3) y_n can be represented as

$$y_n = b_n + b_{n-2} = \begin{cases} +2d, & if\ a_n = a_{n-2} = 1 \\ 0, & if\ a_n \neq a_{n-2} \\ -2d, & if\ a_n = a_{n-2} = 0 \end{cases}$$

Note that the samples b_n are uncorrelated (i.e. either +d for "1" or -d for "0" input). On the other-hand, the samples y_n are correlated (i.e. there are three possible values +2d, 0,-2d depending on a_n and a_{n-2}). It means that the modified duobinary encoding correlates present sample a_n and the previous input sample a_{n-2}.

4) From the diagram, impulse response of the modified duobinary encoder is computed as

$$h(t) = sinc\left(\frac{t}{T}\right) + sinc\left(\frac{t-2T}{T}\right)$$

Decoding Process:

5) The receiver consists of a modified duobinary decoder and a postcoder (inverse of precoder).The modified duobinary decoder implements the following equation (which can be deduced from the equation given under step 3 (see above))

$$\hat{b}_n = y_n - \hat{b}_{n-2}$$

This equation indicates that the decoding process is prone to error propagation as the estimate of present sample relies on the estimate of the previous sample. This error propagation is avoided by using a precoder before modified duobinary encoder at the transmitter and a postcoder after the modified duobinary decoder. The precoder ties the present sample and previous to previous sample (correlates these two samples). The postcoder does the reverse process.

6) The entire process of modified duobinary decoding and the postcoding can be combined together as one algorithm. The following decision rule is used for detecting the original modified duobinary signal samples $\{a_n\}$ from $\{y_n\}$

$$\begin{gathered} if\ y_n < d, \quad then\ \hat{a}_n = 1 \\ if\ y_n > d, \quad then\ \hat{a}_n - 0 \\ if\ y_n = 0,\ randomly\ guess\ \hat{a}_n \end{gathered}$$

Matlab Code:

File 1: modifiedDuoBinaryEncoder.m

```
function [response]=modifiedDuoBinaryEncoder(os_factor)
t=-4:1/os_factor:4; %Limiting the response to -4T to 4T
p=zeros(1,length(t));

for i=1:1:length(t)
    if t(i)==0 p(i)= 1-sin(pi*(t(i)-2))./(pi*(t(i)-2));
    else
        if t(i)==2 p(i)=sin(pi*(t(i)))./(pi*(t(i)))-1;
        else p(i) = sin(pi*(t(i)))./(pi*(t(i)))-sin(pi*(t(i)-2))./(pi*(t(i)-2));
        end
    end
```

```
end
response=p;

figure;
plotHandle=stem(t,response); set(plotHandle,'LineWidth',1.5);
hold on; plotHandle=plot(t,response,'r');
set(plotHandle,'LineWidth',1.5);
title('Impulse Response of modified duobinary encoder');
xlabel('Sampling Time');
%ylabel('Amplitude');
end
```

File 2: precoderD2.m

```
function bn=precoderD2(an)
%an=binary input bits to precoder
bn=zeros(1,length(an));
for i=1:length(an);
    if i==1 bn(i)=xor(an(i),1);
    else
        if i==2 bn(i)=xor(an(i),1);
        else bn(i)=xor(an(i),bn(i-2));
        end
    end
end
bn=[1 1 bn];
end
```

File 3: modifiedDuoBinaryDecoder.m

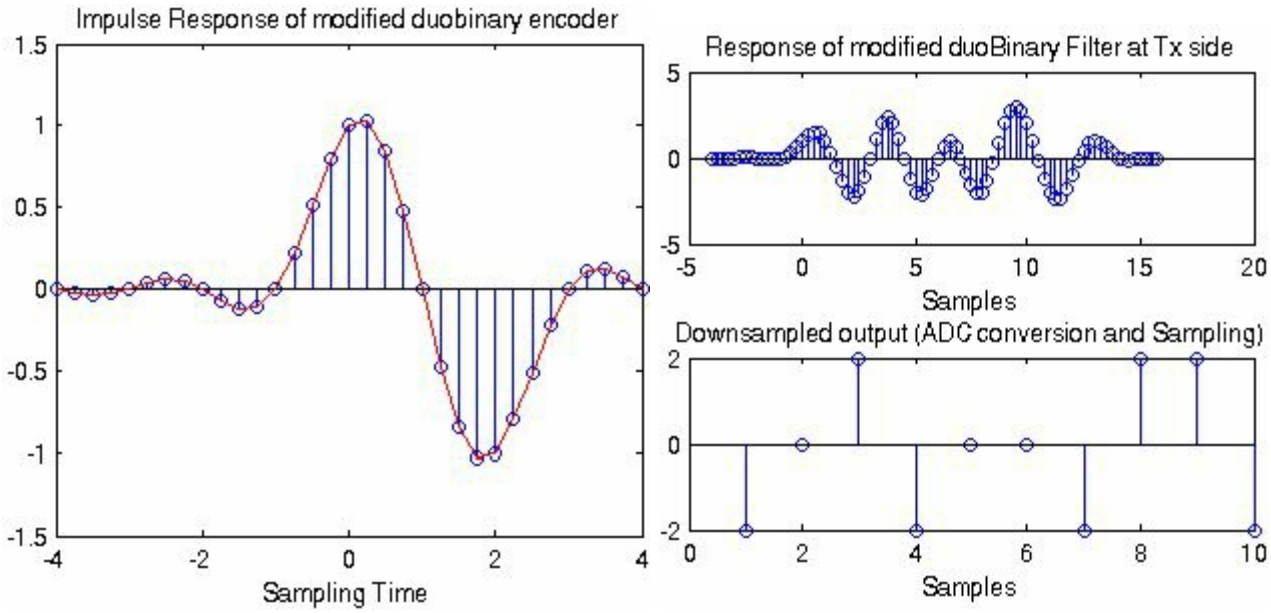
```
function an = modifiedDuoBinaryDecoder(yk)

%Decoding duobinary signal
an=zeros(1,length(yk));
for i=1:length(yk)
    if abs(yk(i))>1  an(i)=1;
    else
        if abs(yk(i))<1 an(i)=0;
        else an(i)=randn(1,1)<=0.5;
        end
    end
end
end
```

File 4: testModifiedDuobinary.m

```
%Simulation of Duobinary Signaling

clc;clear;close all;
overSampling_Factor=4;
```

```
an= [1 0 1 1 0 0 1 1 1 1] %Input binary bits
ak=precoderD2(an) %Precoder
bn=2*ak-1% Level converter

%Duobinary Encoder
Input_bit_os=upsample(bn,overSampling_Factor); %oversampling
pt = modifiedDuobinaryEncoder(overSampling_Factor); % impulse response duobinary encoder
output_of_duoB_filter = conv(Input_bit_os,pt); %convolving bn with duobinary response
L=length(output_of_duoB_filter);
t=-4:1/overSampling_Factor:(-4+(L-1)*1/overSampling_Factor); %defining the response from -4T to
4T

figure;
subplot(2,1,1);
plotHandle=stem(t,output_of_duoB_filter);
set(plotHandle,'LineWidth',1.5);
title('Response of modified duoBinary Filter at Tx side')
xlabel('Samples')
ylabel('Amplitude')

%Simplified algorithm to decoder duobinary signaling
yn=output_of_duoB_filter
midSample=length(-4:1/overSampling_Factor:4)/2;
yn_truncated=yn(midSample+3+overSampling_Factor:end); %Remove unwanted portions(first few
samples till the peak value)

%Now the first sample contains the peak value of the response. From here the samples are extracted
depending on the oversampling factor
yk = downsample(yn_truncated,overSampling_Factor,1); %here offset=1 means starting from 1st
sample
yk=yk(1:length(an));% length output samples  must be equal to length of an. Discard the remaining

%The above mentioned process is equivalent to sampling from the time instant nT=T (i.e. n=1)
subplot(2,1,2);
plotHandle=stem(yk);
set(plotHandle,'LineWidth',1.5);
title('Downsampled output (ADC conversion and Sampling)')
xlabel('Samples')
ylabel('Amplitude');

anEstimated = modifiedDuoBinaryDecoder(yk);
disp('Input Symbols to the system : ');disp(an);
disp('Decoded Symbols :');disp(anEstimated);
```

Simulation Results:



******

Back to Table of Contents

3.4 Raised Cosine Filter

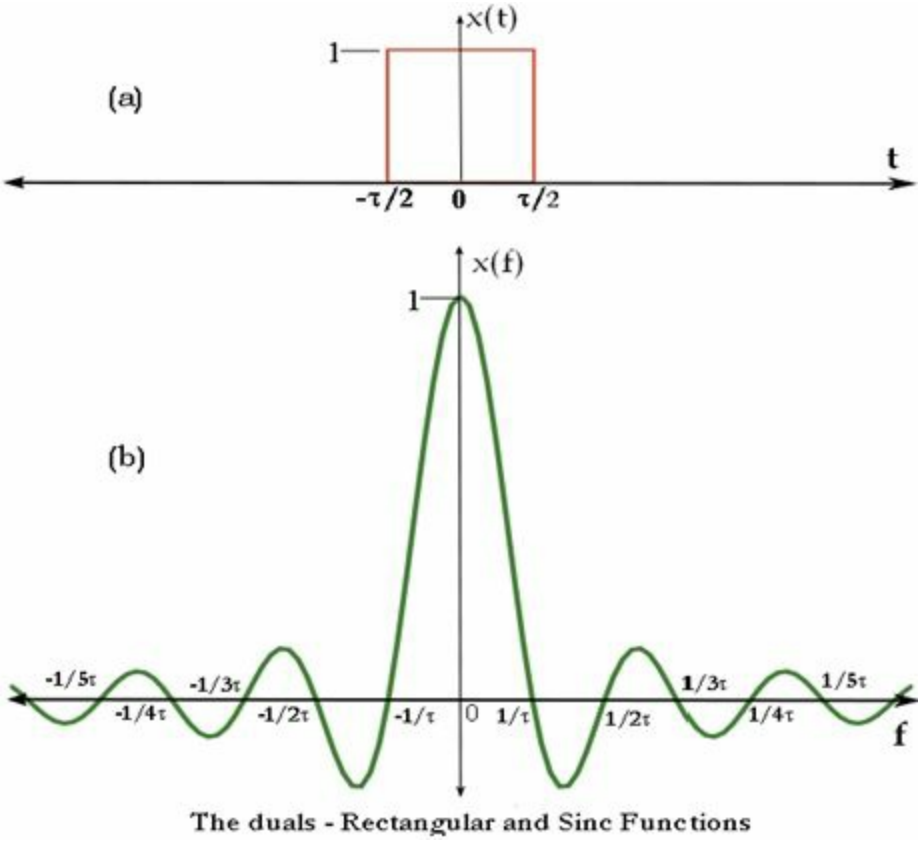
Role of Pulse Shaping in Communication Systems:

In communication systems, data is transmitted as binary bits (ones and zeros). It is easier to implement a binary system using switches, where turning on a switch represents '1' and turning it off represents '0'. Such simple binary systems essentially represent ones and zeros as rectangular pulses of finite duration (say τ seconds).

A rectangular pulse of finite duration τ manifests itself as an infinitely extending sinc pulse in the frequency domain (see figure below). This implies that a rectangular pulse requires infinite bandwidth if it is not to be distorted during its transmission.

Due to the dual nature of Fourier Transform, the following figure is valid if we alternate the frequency and time domain representations. That is, a symbol with finite bandwidth will extend infinitely in time. This implies that to send a single '1' or '0' or a series of them (for a multi-level signaling), you would need infinite time duration. This is absolutely impractical.

In practical terms, signals will not extend infinitely forward and backward in time. But it will definitely be non-zero after the time duration τ. This implies that the residues of adjacent symbols/signals overlap with each other giving rise to Inter Symbol Interference (ISI). If the residual energy from the adjacent symbol is very strong, it becomes impossible to distinguish the present symbol and there is a possibility of it being misinterpreted altogether. To avoid or reduce this effect, "Pulse Shaping" techniques are used to make sure that the data carried by the symbols are not affected by the overlapping effect of adjacent symbols.



The duals - Rectangular and Sinc Functions

In a band-limited system, when we try to increase the data rate, it may lead to Inter Symbol Interference (ISI). There are two criteria that must be satisfied for a non-interference system when pulse shaping is employed.

(1) The pulse shape exhibits a zero crossing at the sampling point of all pulse intervals

(2) The shape of the pulses is such that the amplitude decays rapidly outside of the pulse interval.

A rectangular pulse satisfies the first criterion (where it contains zero crossing – see figure above) but not the second criterion (the energy of the rectangular pulse does not decay rapidly outside the pulse interval and in fact it extends to infinite bandwidth).

Pulse shapes filters like raised cosine filters, square root raised cosine filters and matched filters are employed to shape the transmitted pulses so that they will satisfy the above two criteria of providing an ISI free system. Refer [WebMichael] for a more intuitive treatment of Raised Cosine Filter.

Raised Cosine Filters/Pulses:

A Raised Cosine looks more like a modified sinc pulse in time domain and is given by the following function (This equation is apt for digital domain and Matlab simulation, it is obtained from its analog form by substituting "t" by $n*T_S$)

$$p(t) = \begin{cases} \frac{\pi}{4} sinc\left(\frac{nT_s}{\tau}\right), & if\ \left(1 - \left[\frac{2\alpha nT_s}{\tau}\right]^2\right) = 0 \\ \frac{\cos\left(\frac{\alpha\pi nT_s}{\tau}\right)}{1 - \left(\frac{2\alpha nT_s}{\tau}\right)^2}, & if\ n = 0 \\ \frac{sinc\left(\frac{nT_s}{\tau}\right)\cos\left(\frac{\alpha\pi nT_s}{\tau}\right)}{1 - \left(\frac{2\alpha nT_s}{\tau}\right)^2}, & otherwise \end{cases}$$

Here T_S is the sampling period, n is the sample number, α is a parameter that governs the bandwidth occupied by the pulse and the rate at which the tails of the pulse decay. A value of α = 0 offers the narrowest bandwidth, but the slowest rate of decay in the time domain. When α = 1, the bandwidth is 1/τ, but the time domain tails decay rapidly. For more details on investigation of raised cosine characteristics and its implementation – refer links - [WebMichael], [WebKen], [WebComplex2Real] mentioned under the "External Links" section at the end of this book.

The following Matlab program simulates a Raised Cosine Filter and plots its frequency response.

Matlab Code:

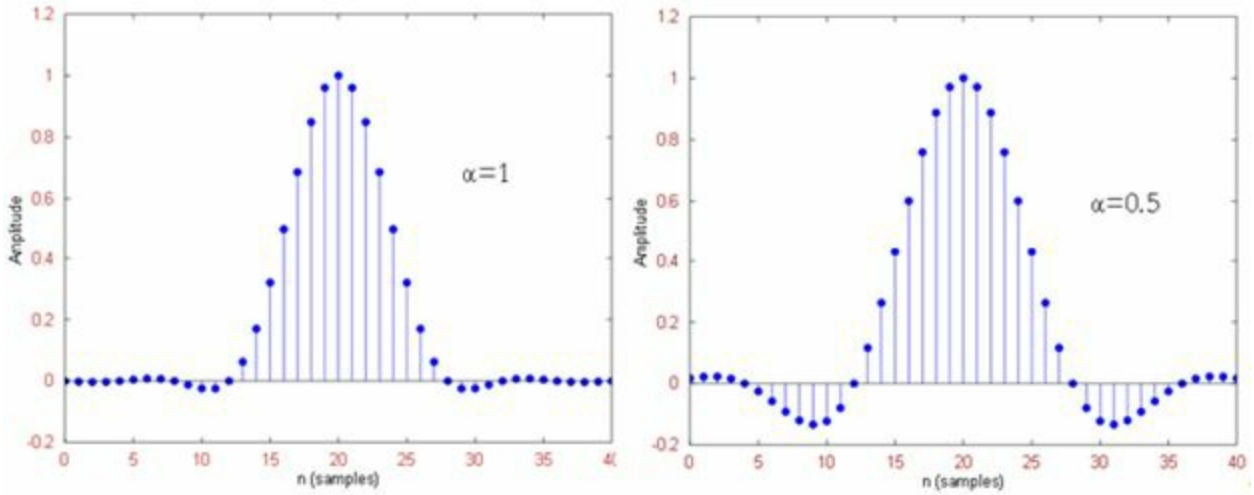
File1: raisedCosineFilter.m

```
%Raised Cosine Filter
L=41; %Filter Length
R=1E6; %Data Rate = 1Mbps
Fs=8*R; %Oversampling by 8
T=1/R;
Ts=1/Fs;
alpha =0.5; % Design Factor for Raised Cosing Filter

%-------------------------------------------------------
%Raised Cosing Filter Design
%-------------------------------------------------------
if mod(L,2)==0
    M=L/2 ; % for even value of L
else
    M=(L-1)/2; % for odd value of L
end
g=zeros(1,L); %Place holder for RC filter's transfer function

for n=-M:M
    num=sin(pi*n*Ts/T)*cos(alpha*pi*n*Ts/T);
    den=(pi*n*Ts/T)*(1-(2*alpha*n*Ts/T)^2);
    g(n+M+1)=num/den;
    if (1-(2*alpha*n*Ts/T)^2)==0
        g(n+M+1)=pi/4*sin(pi*n*Ts/T)/(pi*n*Ts/T);
    end
```

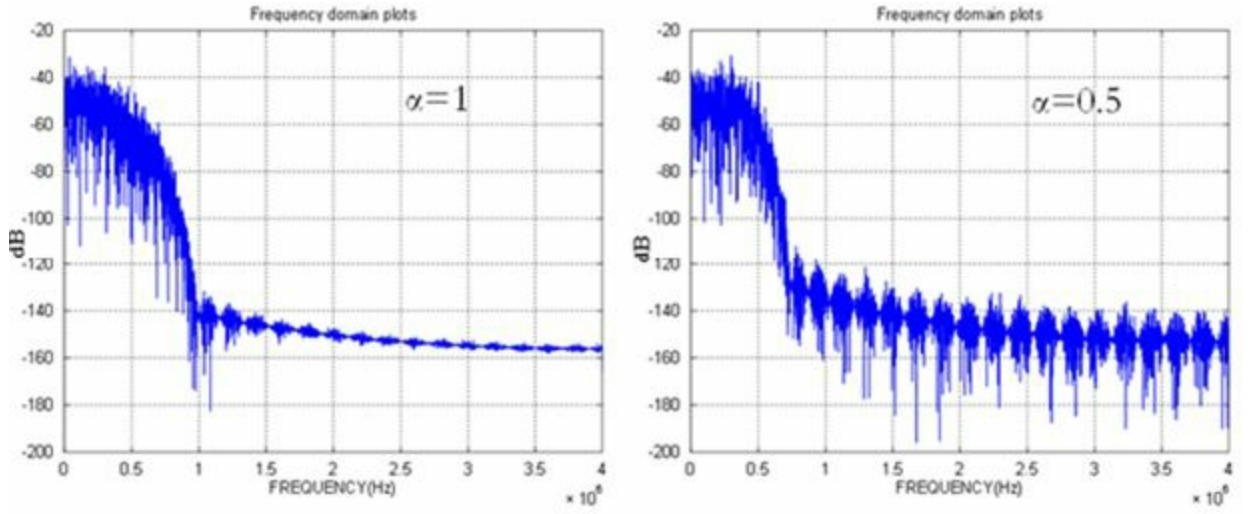
```
    if n==0
        g(n+M+1)=cos(alpha*pi*n*Ts/T)/(1-(2*alpha*n*Ts/T)^2);
    end
end

%-------------------------------------------------------
% Plot the transfer function of RC filter
figure;
impz(g,1);

%-------------------------------------------------------
%Generate data of random 1s and 0s
data=2*(rand(1,1000)>=0.5)-1; %Polar encoding : 1= +1V, 0=-1V
output=upsample(data,Fs/R);
%y=conv(g,output); %Convolving the data signal with the Raised Cosine Filter
y=filter(g,1,output); %you can use either Conv function or filter function to obtain the output

%-------------------------------------------------------
%Plot data and RC filtered Output
figure;
subplot(2,1,1);
stem(data);
title('Input data to the Raised Cosine Filter');
xlabel('Samples');
ylabel('Amplitude');
axis([0,20,-1.5,1.5])
subplot(2,1,2);
plot(y);
axis([0,160,-1.5,1.5])
title('Response of the Raised Cosine Filter for the given Input');
xlabel('Samples');
ylabel('Amplitude');

% Obtain FFT of the output to plot its frequency response.
Fn=Fs/2;
NFFY=2.^(ceil(log(length(y))/log(2)));
FFTY=fft(y,NFFY);%pad with zeros
NumUniquePts=ceil((NFFY+1)/2);
FFTY=FFTY(1:NumUniquePts);
MY=abs(FFTY);
MY=MY*2;
MY(1)=MY(1)/2;
MY(length(MY))=MY(length(MY))/2;
MY=MY/length(y);
f1=(0:NumUniquePts-1)*2*Fn/NFFY;
```

```
%Plot Frequency spectrum
figure;
plot(f1,20*log10(abs(MY).^2));xlabel('FREQUENCY(Hz)');ylabel('DB');
grid
title('Frequency domain plots')
```

The RC filtered outputs for α = 1 and α =0.5 are plotted next. As we can see that as α increases (from 0 to 1), the pulse duration decreases (in frequency domain it will consume excess bandwidth). We must strike an optimum balance between the value of α and the bandwidth it occupies, for the pulse to travel undistorted in a given band-limited channel.



The frequency response for the Raised Cosine Filtered output for α =1 and α =0.5 are plotted below for comparison. The frequency response shows minimized side lobes for α =1, whereas for the case α =0.5 the side lobes' energy is higher when compared to α =1. Also, for α =1, the eye opening is clearer when compared to α =0.5. This implies that as we increase the value of α from 0 to 1, the ISI decreases whereas the excess bandwidth of the signal increases. An optimum balance between the value of α and the bandwidth of the band-limited channel is necessary for ISI free reliable communication between the transmitter and receiver.



******

Back to Table of Contents

3.5 Square Root Raised Cosine Filter (Matched/split filter implementation)

An implementation of a pulse shaping filter namely Raised Cosine Filter was discussed in the previous chapter.

The code was modified to implement a Square Root Raised Cosine Filter (SRRC). Instead of using a single Raised Cosine filter at the transmitter, a square root raised cosine filter is used at both transmitter and receiver. Using a single raised cosine filter at the transmitter or two separate SRRC filters (one at the transmitter and another at the receiver) will have the same effect on ISI. But using SRRC filters at the transmitter and receiver provides better matching characteristics and thereby improves the overall SNR of the system considerably.

The frequency response of the SRRC filter is the square root of the frequency response of the raised-cosine filter. The impulse response of the SRRC filter (according to 3GPP standard [3GPPTS25104]) is given by

$$p(t) = \frac{\sin\left(\pi\frac{t}{T}[1-\alpha]\right) + 4\alpha\frac{t}{T}\cos\left(\pi\frac{t}{T}[1+\alpha]\right)}{\pi\frac{t}{T}\left(1-\left[4\alpha\frac{t}{T}\right]^2\right)}$$

With the following values at t=0 and $t/T = \pm 1/(4\alpha)$

$$p(t)|_{t=0} = (1-\alpha) + \frac{4\alpha}{\pi}$$

$$p(t)|_{\frac{t}{T}=\pm\frac{1}{4\alpha}} = \frac{\alpha}{\sqrt{2}}\left[\left(1+\frac{2}{\pi}\right)\sin\left\{\frac{\pi}{4\alpha}\right\} + \left(1-\frac{2}{\pi}\right)\cos\left\{\frac{\pi}{4\alpha}\right\}\right]$$

There exists several forms of equations to implement SRRC filters.

Matlab function to implement Square Root Raised Cosine Filter:

```
function [response]=srrc(os_factor,roll_off)
a=roll_off;
t=-4:1/os_factor:4; %Limiting the response to -4T to 4T

%This can be increased or decreased according to the requirement
p=zeros(1,length(t));
for i=1:1:length(t)
    if t(i)==0 p(i)= (1-a)+4*a/pi;
    else
        if t(i)==1/(4*a) || t(i)==-1/(4*a) p(i)=a/sqrt(2)*((1+2/pi)*sin(pi/(4*a)+(1-
        2/pi)*cos(pi/(4*a))));
        else p(i) = (sin(pi*t(i)*(1-a))+4*a*t(i).*cos(pi*t(i)*(1+a))./(pi*t(i).*(1-(4*a*t(i)).^2));
        end
```

```
    end
end
response=p./sqrt(sum(p.^2)); %Normalization to unit energy
end
```

Simulation of Square Root Raised Cosine Filter in Matlab:

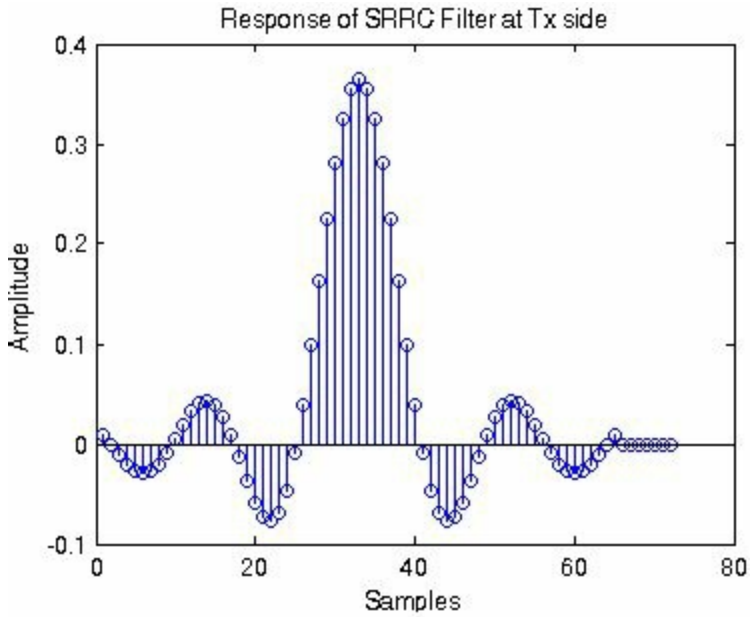
Define oversampling factor (to simulated digital to analog conversion in the transmitter), input bits to be transmitted and the impulse response of the desired SRRC filter (as shown in the equation/function above).

The SRRC filter's impulse response is computed for t=-4T to 4T duration. This duration can be changed to any appropriate desired value.

```
clc;clear;close all;
overSampling_Factor=8;
Input_bit = [1]; %Bits to be transmitted
Input_bit_os=upsample(Input_bit,overSampling_Factor); %oversampling
alpha=0.1; % roll-off factor of Root Raised Cosine Filter
pt = srrc(overSampling_Factor,alpha); % impulse response of SRRC filter
```

Convolve the impulse response of the root raised cosine filter with the oversampled version of input bits and plot the resulting response. This is the response of the filter for the input bits at the transmitter side.

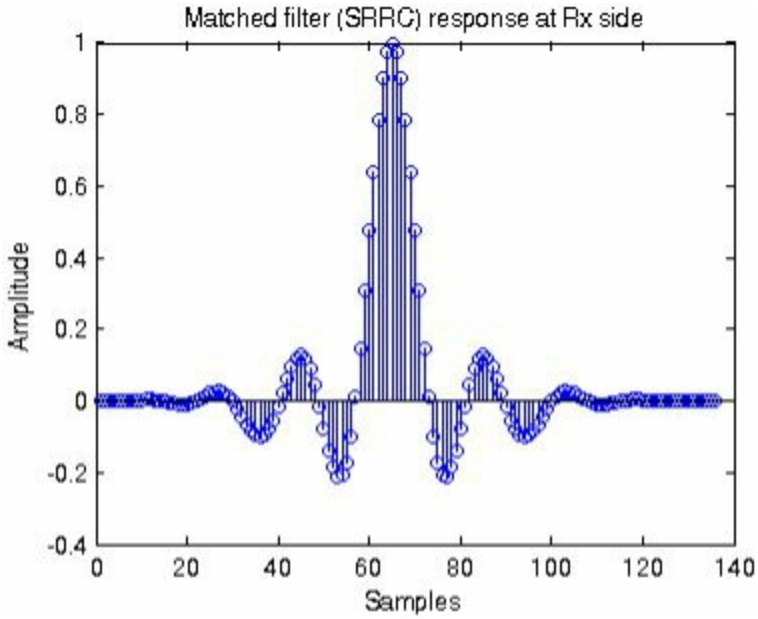
```
output_of_srrc_filter = conv(Input_bit_os,pt);
stem(output_of_srrc_filter);
title('Response of SRRC Filter at Tx side')
xlabel('Samples')
ylabel('Amplitude')
```



At the receiver side the received signal ("output_of_srrc_filter") is convolved with a matched filter (which is same as the transmitting filter).Note that no noise is added to the incoming signal. If you wish, you could add some noise to the signal before performing the matched filter operation.

```
%Receiver side; using a matched filter (that is matched to the SRRC pulse in the transmitter)
```

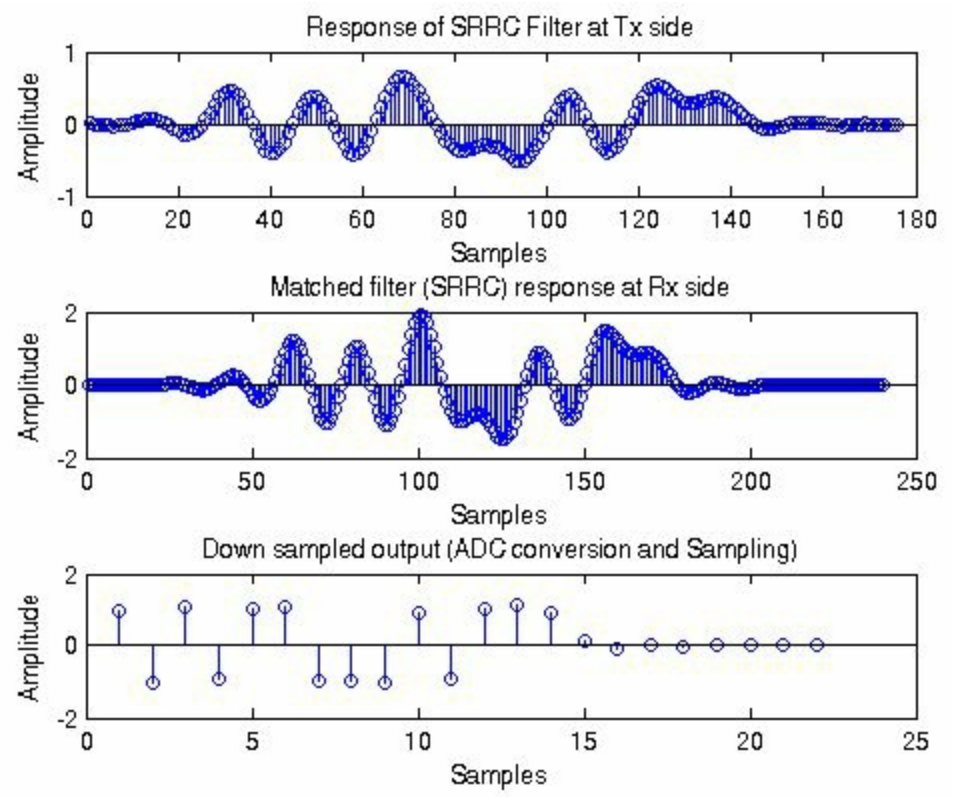
```
y = conv(output_of_srrc_filter,pt);
stem(y);
title('Matched filter (SRRC) response at Rx side');
xlabel('Samples');
ylabel('Amplitude');
```



Note that the peak value of the matched filter response occurs at 65th sample (this number will vary depending on the oversampling rate). Noting down the sample number of the peak value for a single bit input at the transmitter is important in downsampling operation (equivalent to recovering the input bits by sampling using a Digital to Analog Converter at the receiver). A small snippet of code is given below, to extract the recovered bits out of the matched filter response.

```
midSample=length(-4:1/overSampling_Factor:4);
y_truncated=y(midSample-1:end); %Remove unwanted portions(first few samples till the peak value)
%Now the first sample contains the peak value of the response. From here the samples are extracted depending on the oversampling factor

y_down = downsample(y_truncated,overSampling_Factor,1); %here offset=1 means starting from 1st sample %retain every 8th sample

stem(y_down);
title('Down sampled output (ADC conversion and Sampling)');
xlabel('Samples');
ylabel('Amplitude');
```

Following simulation results are shown for a series of input bits: Input_bit = [1 -1 1 -1 1 1 -1 -1 -1 1 -1 1 1 1]. It is interesting to note how the effect of inter symbol interference is mitigated by the Square Root Raised Cosine Filters at both the transmitter and receiver. The roll-factor of the SRRC filter can be adjusted to see the effects of ISI on the recovered samples.



******

Back to Table of Contents

3.6 Gibbs Phenomena – A demonstration

In signal processing, brick wall characteristics in frequency domain are desired for most of the filters. For example, a typical ideal low pass filter necessitates sharp transition between passband and stopband. But in the real world it is impossible to have such ideal characteristics. In practice, filters are designed to have characteristics close to ideal (but they can never reach the ideality).

An ideal brick wall filter (rectangular function with abrupt transition) in frequency domain translates to a sinc (sin(x)/x) function of infinite duration in time domain. In actual filter design, this sinc function in time domain is truncated appropriately because it is not possible to have infinite time duration signal in reality.

(1)This truncation of an infinite duration signal in time domain leads to a phenomenon called Gibbs phenomenon in frequency domain.

(2)If truncation is done in the frequency domain it can also be observed in the time domain.

Since some of the samples in time domain (harmonics in frequency domain for case 2 given above) are not used in the reconstruction, it leads to oscillations and ringing effect in the other domain. This effect is called Gibbs phenomenon.

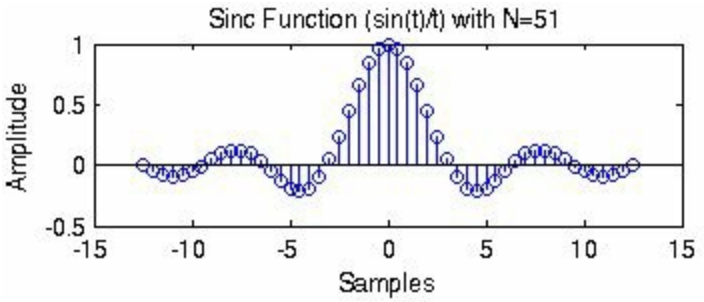
This effect, which is due to abrupt discontinuities, will be significant no matter how long is the number of samples (used to construct frequency domain response). The situation can be improved by using smoothly tapered windowing techniques like Blackman, Hamming, Hanning, Keiser windows

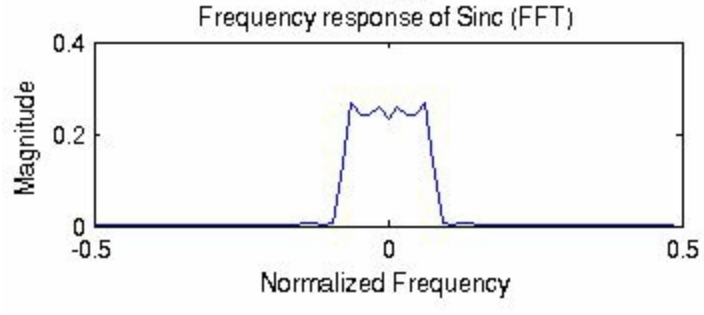
etc.

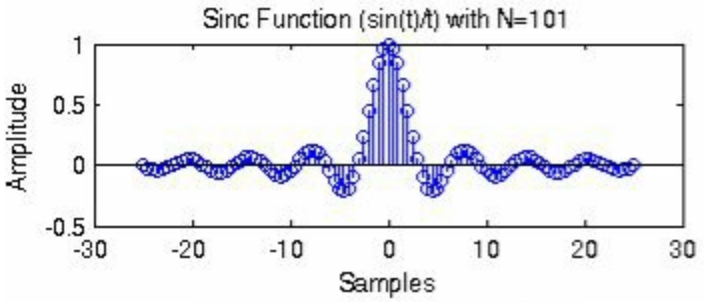
Let us demonstrate the Gibbs phenomenon using Matlab. The simulated output plot shows that, as the sample size taken for FFT increases, the frequency response approaches ideal brick wall response but with increasing ringing effect on its edges.

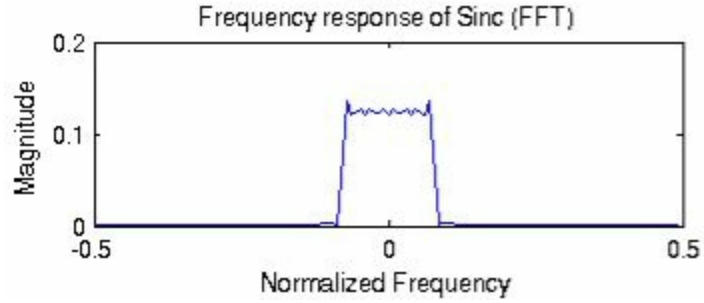
Matlab Code:

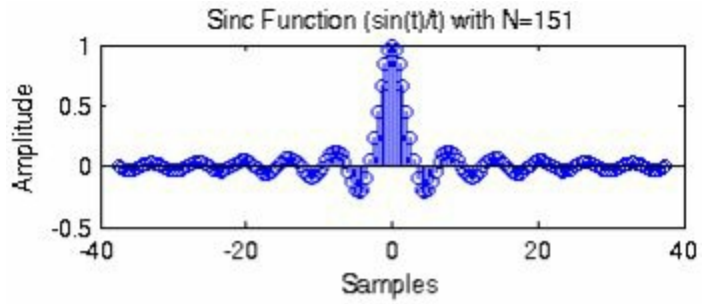
```
%Gibbs Phenomenon

clc; % clear command window
clear; % clear all stored variables
N=51:50:501; %Number of points in each sinc Function
for i=N,
    max=(0.5*(i-1))/2;
    time = -max:0.5:max;
    disp(length(time));

    %Sinc Function
    x= sin(time)./time;
    x(isnan(x))=1; % Special case to handle sinc(0)
    subplot(2,1,1);
    stem(time,x);
    title(['Sinc Function (sin(t)/t) with N=',num2str(i)]);
    xlabel('Samples');
    ylabel('Amplitude');

    %Take FFT
    L=length(x);
    NFFT = 2^nextpow2(L); % Next power of 2 from length of x
    X = fft(x,NFFT)/L;
    X = fftshift(X);
    f = (-NFFT/2:NFFT/2-1)/NFFT;

    subplot(2,1,2);
    plot(f,2*abs(X)) ;
    title('Frequency response of Sinc (FFT)');
    xlabel('Normalized Frequency');
    ylabel('Magnitude');
    pause;% wait for user input to continue
end
```

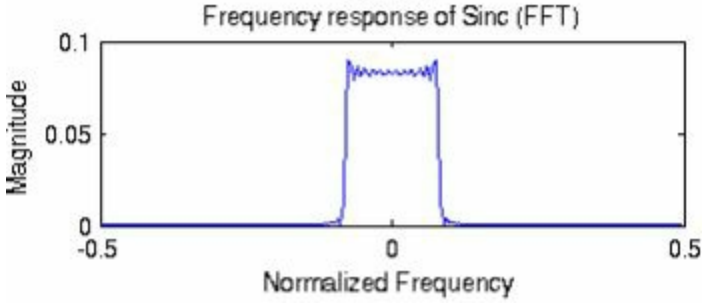












******

Back to Table of Contents

3.7 Moving Average (MA) Filter

The moving average filter is a simple Low Pass FIR (Finite Impulse Response) filter commonly used for smoothing an array of sampled data/signal. It takes M samples of input at a time and takes the average of those M-samples and produces a single output sample. It is a very simple LPF (Low Pass Filter) structure that comes handy for scientists and engineers to filter unwanted noisy components from the intended data.

As the filter length increases, (indicated by the parameter M) the smoothness of the output also increases, whereas the sharp transitions in the data are made increasingly blunt. This implies that this filter has excellent time domain response but a poor frequency response.

The MA filter performs three important functions:

1) It takes M input points, computes the average of those M-points and produces a single output point

2) Due to the computation/calculations involved, the filter introduces a definite amount of propagation delay

3) The filter acts as a Low Pass Filter (with poor frequency domain response and a good time domain response).

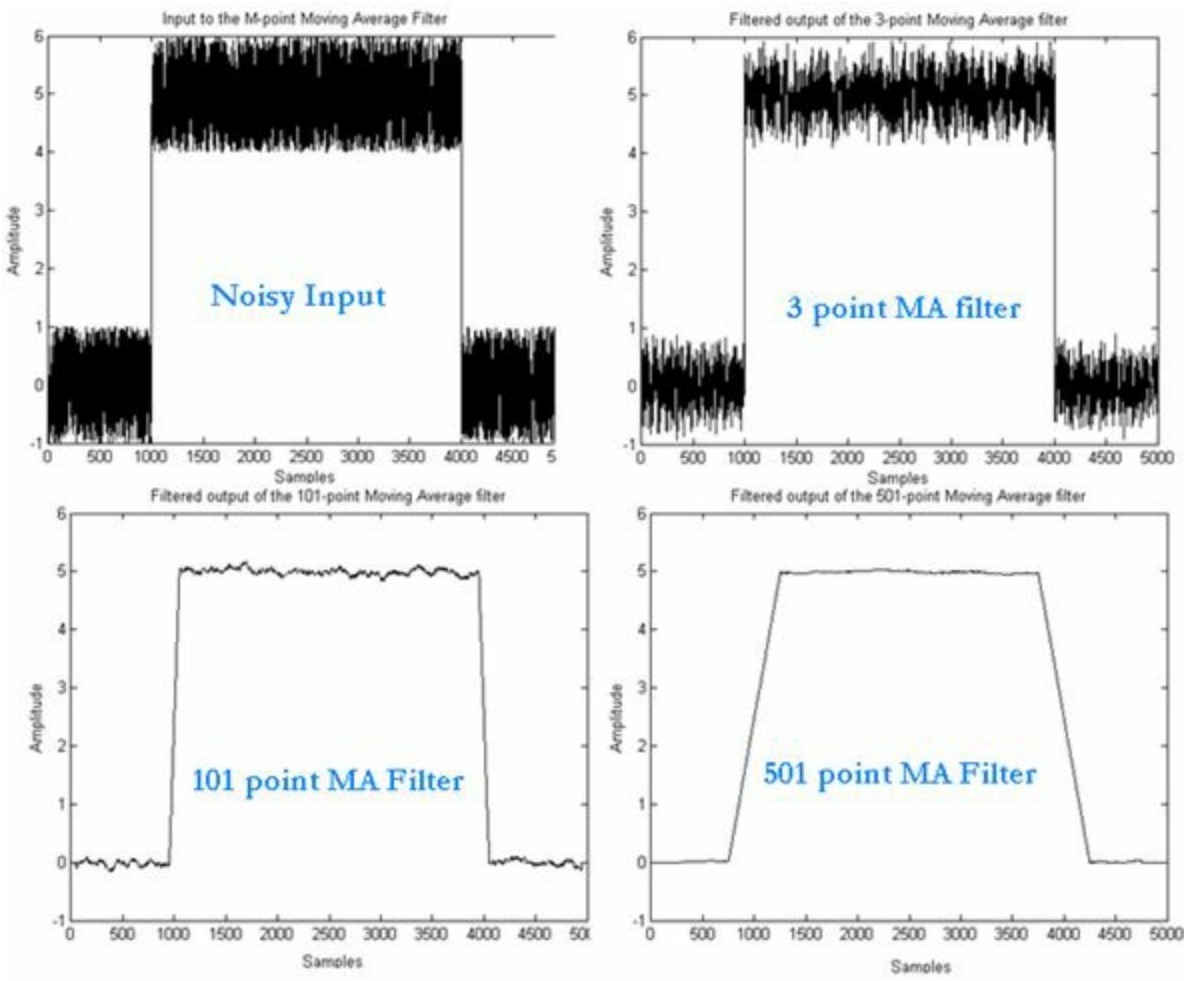
Matlab Code:

Following Matlab code simulates the time domain response of an M-point Moving Average filter and also plots the frequency response for various filter lengths.

```
%Moving Average Filter (MA) Demonstration
%Computes and plots the time domain output of a M-point Moving Average Filter. Note : The parameter M must be an odd number in order to realize the filter.
%Plots the frequency response of the M-point Moving Average filter for comparison.
clear;
clc;
M=51; %Number of taps/points in the MA filter

data=[zeros(1,1000),5*ones(1,3000),zeros(1,1000)]; %5000 points sample data
noise= 2*rand(1,5000)-1;
x=data+noise; %Adding some noise to the data
%Plot the input

figure;
plot(x);
xlabel('Samples');
ylabel('Amplitude');
title('Input to the M-point Moving Average Filter');
%Computing the output response of the given moving average filter

figure;
```

```
y=zeros(1,5000);
for i=(M-1)/2+1:length(data)-(M-1)/2
    y(i)=0;
    for k=-(M-1)/2:(M-1)/2
        y(i)=y(i)+x(i+k);
    end
    y(i)=y(i)/M;
end
plot(y);
xlabel('Samples');
ylabel('Amplitude');
title(['Filtered output of the ',num2str(M),'-point Moving Average filter']);

%Frequency Response for various Filter lengths
colors=['r','b','k','g']; %To plot in different colors
colorIndex=1;

figure;
M = [3,11,51,101];
for i=M
b=repmat(1/i,1,i);
a=1; %Since this is a FIR filter a=1 and b=[1/M,1/M,1/M,...,repeated M times]
[h,w]=freqz(b,a); %Using the Freqz function to get the frequency response
h=abs(h);
w=w/max(w);
plot(w,h,colors(colorIndex)); %Plotting the response
colorIndex=colorIndex+1;
hold on;
end
hold off;

legend('M=3','M=11','M=51','M=101');
xlabel('Normalized Frequency');
ylabel('Magnitude');
title('Frequency Response of M-point Moving average filters');
```

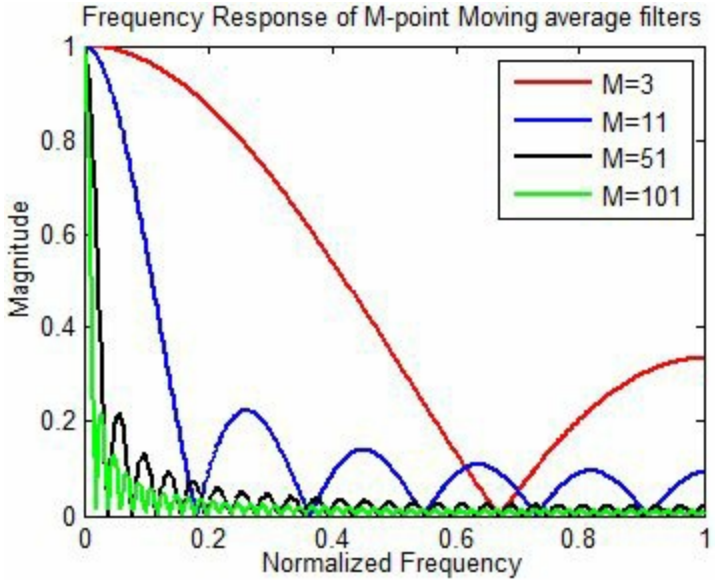
Time Domain Response:



The first plot shows the input to the moving average filter. The input is noisy and our objective is to reduce the noise. The next figure is the output response of a 3-point Moving Average filter. It can be deduced from the figure that the 3-point Moving Average filter has not done much in filtering out the noise. We increase the taps further to 101 and 501 and we can observe that even-though the noise is almost zero, the transitions are blunted out drastically (observe the slope on either side of the signal and compare them with the ideal brick wall transition in the input).

Frequency Response:

From the frequency response it can be asserted that the roll-off is very slow and the stop band attenuation is not good. Given this stop band attenuation, clearly, the moving average filter cannot separate one band of frequencies from another. As we know a good performance in the time domain results in poor performance in the frequency domain, and vice versa. In short, the moving average is an exceptionally good smoothing filter (the action in the time domain), but an exceptionally bad low-pass filter (the action in the frequency domain)



******

Back to Table of Contents

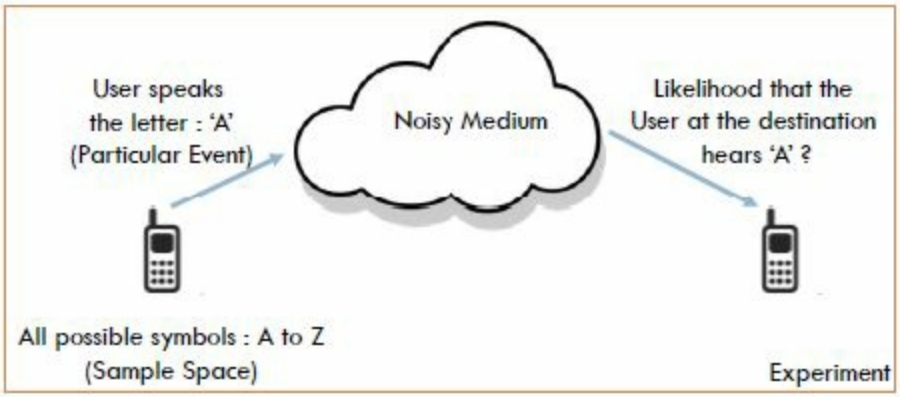
Chapter 4: Probability and Random Process

4.1 Introduction to concepts in probability

What is Probability?

Probability is a branch of mathematics that deals with uncertainty. The term "probability" is used to quantify the degree of belief or confidence that something is true (or false). It gives us the likelihood of occurrence of a given event. It is expressed as a number that could take any value in the closed interval [0,1]

Consider the following experiment describing a simple communication system. A user transmits data through a noisy medium and another user receives it. Here, the sender utters a single alphabet on the phone. Due to the noise characteristics of the communication medium, we do not know whether the user at the destination will be able to hear what the sender has already spoken. Before performing the experiment, we would like to know the likelihood that the user at the destination hears the particular syllable (given the noise characteristics). This likelihood of the particular event is called probability of the event.

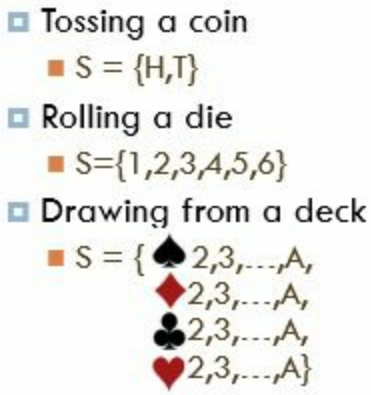


Experiment:

Any activity that can produce observable results is called an experiment. For example, tossing a coin (observable results: Head/Tail), Rolling a die (observable results: numbers on the faces of the die), drawing a card from a deck (observable results: symbols, numbers and alphabets on the cards), sending & receiving bits in a communication system (observable results: bits/alphabets transferred or voltage level at the receiver).

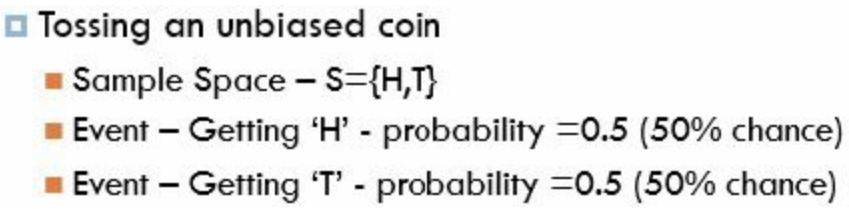
Sample Space:

Given an experiment, the sample space comprises a set of all possible outcomes of the experiment. It plays the role of the universal set when modeling the experiment. It is denoted by the letter - 'S'. Following examples illustrate the sample spaces for various experiments.



Event:

It is also a set of outcomes of an experiment. It is a subset of sample space. Each time the experiment is run, either a particular event occurs or it does not occur. Events are associated with a probability number.



Types of Events:

Events can be classified according to their relationship with one another. The following table shows the classification of events and their definition.

Independent events	Dependent events	Mutually exclusive events
Events are not affected by each other	Events are affected by each other	Events cannot occur at the same time
Example: Tossing a coin three times -outcome of one toss will not affect the other –tosses are totally independent	Examples: Continuously drawing from a deck of cards without replacing the card Counting the number of heads on second toss	Example: Tossing a coin, occurrences of head or tail are mutually exclusive

Computing Probability:

The proability of the occurrence of an event (say 'A') is given by the ratio of number of ways that particular event can happen and the total number of all possible outcomes.

$$P(A) = \frac{Number\ ways\ an\ event\ can\ happen}{Total\ number\ of\ all\ possible\ outcomes} = \frac{n(A)}{n(S)}$$

For example, consider the experiment of an unbiased rolling of a die. The sample space is given by S={1,2,3,4,5,6}. Let's say that an event is defined as getting '4' when you roll the die. The probability of getting the face with '4' (event) can be calculated as follows.

$$P(A) = \frac{n(A)}{n(S)} = \frac{1}{6}$$

Axioms of Probability:

Following definitions are assumed for the axioms listed below: 'S' denotes the sample space of an experiment, 'A' and 'B' are events and P(A) denotes the probability of occurrence of event 'A'.

Axioms :

- $P(A) \geq 0$ (Probability cannot be negative)
- $P(S) = 1$ (Probability of everything is 1)
- If the events A & B are mutually exclusive, then
  - $P(A\ and\ B) = P(A \cap B) = \emptyset$
    - Probability of occurrence of both events is null
  - $P(A\ or\ B) = P(A \cup B) = P(A) + P(B)$
    - Probability of occurrence of either of the events

Properties of Probability:

The definition of probability - has some properties as listed below.

1) $0 \leq P(A) \leq 1$
2) $P(\bar{A}) = 1 - P(A)$
3) $P(\emptyset) = 0$
4) $If\ A \subseteq B,\ P(A) \leq P(B)$
5) Generally, for any two events
   - $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
6) If the two events are mutually exclusive
   - $P(A \cup B) = P(A) + P(B)$
7) If the two events are independent
   - $P(A \cap B) = P(A) \times P(B)$

Here the symbol Ø indicates null event, Ā indicates that the event A is NOT occuring.

Joint probability and Marginal probability:

Joint probability is defined as the probability that two or more events occur simultaneously. For two events A and B, the joint probability is denoted by P(A,B) or P(A∩B).

Given two or more events, the marginal probability is the probability of occurrence of a single event. It is also called a-priori probability.

The following table illustrates the concept of computing the joint and marginal probabilities. Here, four events (P, Q, R, S) are used for illustration. For example, the table indicates that the probability of occurrence of both events R & Q is given by b/n. This is the joint probability of R and Q. Adding all the probabilities either row wise or column wise gives us the marginal probability of a single event. For example, adding a/n and b/n gives the marginal probability of event(???) similarly, adding a/n and c/n gives the marginal probability of event P.

Joint Probability of R & Q

Probabilities	Event P	Event Q	Total
Event R	a/n	b/n	(a+b)/n
Event S	c/n	d/n	(c+d)/n
Total	(a+c)/n	(b+d)/n	1

Marginal Probability of P

$$P(Q) = P(R, Q) + P(S, Q) = P[(R \cap Q) \cup P(S \cap Q)]$$

Conditional probability or Posteriori probability:

Conditional probabilities (also called posteriori probability) deal with dependent events. It is used to calculate the probability of an event given that some other event has already occurred.

It is denoted as P(B|A)–meaning that 'the probability of event B given that the event A has occurred already'. It is called "a-posteriori" because it is only available "after" observing A (the first event).

The conditional probability P(B|A) is mathematically computed as

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

******

Back to Table of Contents

4.2 Bayes' Theorem

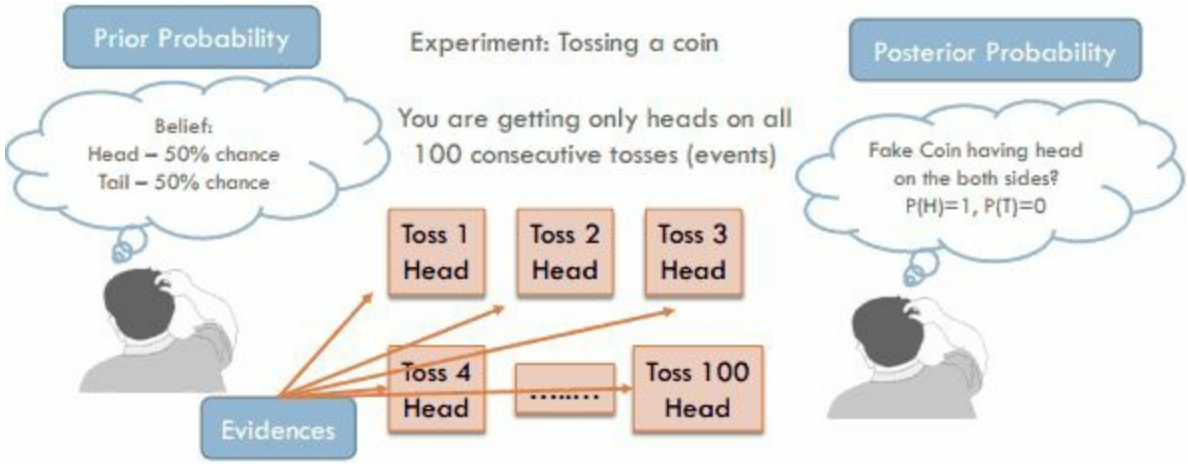
In statistics, the process of drawing conclusions from data subject to random variations – is called "statistical inference". Usually, in any random experiment, the observations are recorded and conclusions have to be drawn based on the recorded data set. Conclusions over the underlying random process are necesseray to establish one or many of the following:

* Estimation of a parameter of interest (For example: the carrier frequency estimation in the receiver)
* Confidence and Credibility of the estimate
* Rejecting a preconceived hypothesis
* Classification of data set into groups

Several schools of statistical inference have evolved over time. Bayesian interference is one of them.

Thought Experiment:

Consider the following thought experiment.



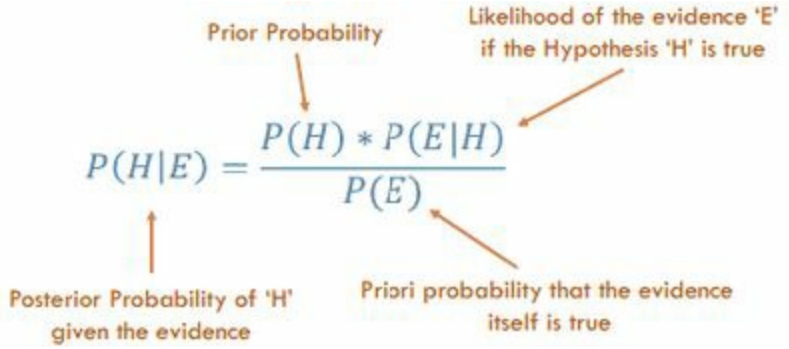
You are asked to conduct a random experiment with a given coin. You are told that the coin is unbiased (Probability of obtaining head or tail is equal and is 50%). You believe (before conducting the experiment) that the coin is unbiased and that the chance of getting head or tail is indeed equal to be 0.5 (50%).

Assume that you have not looked at both sides of the coin and simply you start to conduct the experiment. You start to toss the coin repeatedly and record the events (This is the observed data). On the first toss you observe the coin lands on the ground with head faced up. On the second toss, again the head shows up. On subsequent tosses, the coin always shows up head. You have tossed 100 times and all these tosses you observe only head. Now what will you think about the coin? You will really start to doubt if the coin is unbiased (probably a fake coin) and start to think that both sides of the coin is engraved with "head" (No tail engraved on the coin). Now, based on the evidences (observed datasets), you will change your belief. At the end of the experiment, your belief about the "unbiasedness" of the coin has changed.

This is what Bayes' theorem or Bayesian Inference is all about. It is a general principle about learning from experience. It connects beliefs (called prior probability in statistics) and evidences (observed data). Based on the evidence, the degree of belief is refined. The degree of belief after conducting the experiment is called posterior probability.

Bayes' theorem is sometimes applied iteratively, (as in LDPC decoding with soft decisions), where the prior probabilities (beliefs) are refined iteratively. Each iteration begins with a prior-probability, and after obtaining the data from the random experiment, the posterior probability is recorded. For the next iteration, the posterior probability from the previous iteration becomes the prior probability of the next iteration and the process continues till a pre-defined number of iterations.In this way, the degree of belief on some parameter is refined iteratively and the conclusions are drawn.

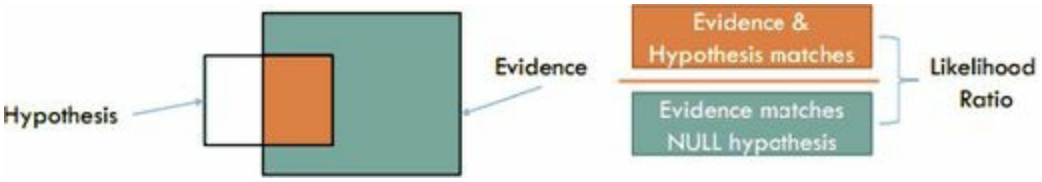
The following equation summarizes Bayes' theorem



Here, H represents any hypothesis (example: the belief that the coin in the above experiment is unbiased), E – represents the evidence. Therefore, P(H) represents the prior probability (degree of belief about the hypothesis before conducting the experiment). P(H|E), is a conditional probability that represents the posterior probability. It is the adjusted degree of belief based on the evidences observed. P(E) represents the probability that the evidence itself is true. P(E|H) represents the probability that the given evidence has occurred because of the manifestation of the given hypothesis – that is, the likelihood that the occurance of the given evidence is because the hypothesis is true.

Refer to the excellent YouTube video - [NatNapoletano] (see reference section at the end of this ebook) for knowing more about Bayes' theorem.

Consider the diagram on the left side of the following figure. We have an evidence (represented by the green box – Actually some portion of the green box is overlapped under the orange box). The observed-evidence may or may not match the given hypothesis completely. The orange box represents the region where evidence has occured because of the hypothesis.The remaining portion of the green box represents the likelihood that the evidence does not match the hypothesis (otherwise called "Null hypothesis" in statistical terms) The ratio of these two regions (orange and non overlapping portion green box) is called likelihood ratio. Likelihood ratios are used in model selection.



******

Back to Table of Contents

4.3 Distributions and Density Functions

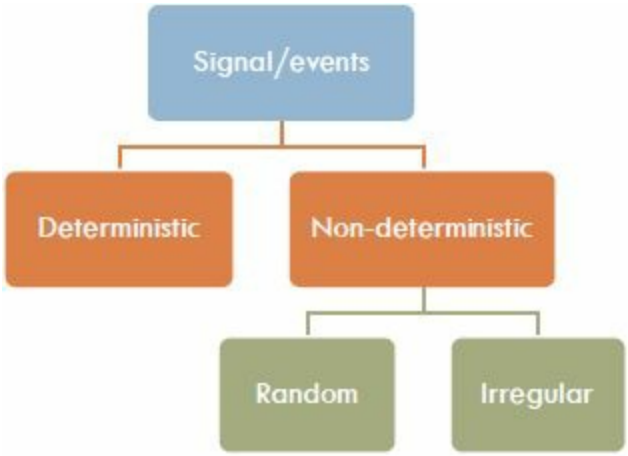
Deterministic Vs Non-deterministic events:

Signals or events can be classified as deterministic or non-deterministic signals/events. Deterministic events/signals can be expressed by mathematical functions and are identically reproducible with repeated measurements. On the other hand, non-deterministic events are not deterministic in nature. The non-deterministic events are further classified into random and irregular events.

Consider the following equation describing a wave using amplitude (A), angular frequency (ω), time (t) and phase (θ)

$$V = A\ sin(\omega t + \theta) \quad ;\ A, \theta\ \omega, t \in \mathbb{R}$$

Here the ampltitude, phase, angular frequency and time take values from the real domain.The resulting wave (V) is predictable in nature and is repeatable.



Random events:

Random events/signals are not predictable or repeatable. They are used to model real events that arise from unknown and non-ascertainable influences. They are statistically describable. Outcome of a roulette game, rolling a dice, tossing a coin, bio-medical signals, life expectancy of an engine and noise in data/speech transmission are some examples of random events.

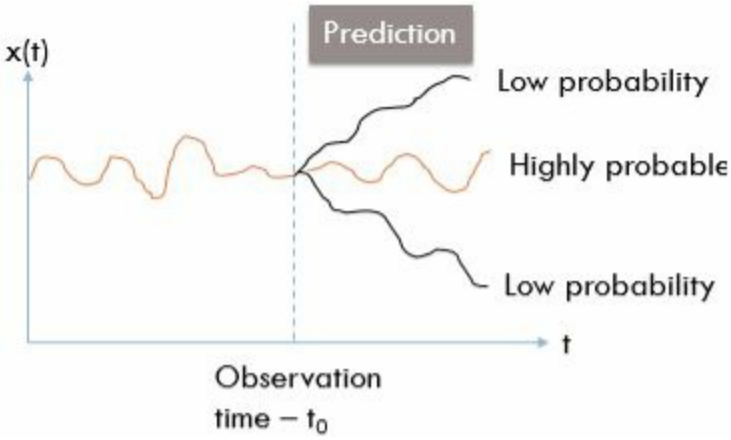
Consider the following equation describing a wave using amplitude (A), angular frequency (ω), time (t) and phase (θ)

$$V = A\ sin(\omega t + \theta) \quad ;\ \omega, t \in \mathbb{R}\ , A\ and\ \theta\ are\ random$$

Here the angular frequency and time takes values from the real domain. The amplitude and phase are random. Since the amplitude and phase are random, the resulting wave (V) is also random in nature and is unpredictable. But it can be described statistically using the statistical properties of amplitude and phase.

Let us say we have observed a signal till some time t_0. We would like to predict the future course of the signal using the observations done till the time instant t_0. If the signal is random, we can make

probabilistic statements with regards to prediction. Referring to the figure below, we can immediately say that (the middle curve in the prediction region) is highly probable than the other two curves in the region. Thus, as far as random signals are concerned, they cannot be predicted in particular, but we can make probabilistic statements about its future values.



Irregular Signals/Events:

Irregular signals are not describable even by statistical or probabilistic terms. They do not follow any rules or laws, therefore neither predictable nor describable. They are just irregular in nature.

The due course of this literature deals only with random signals/events.

Random Variable:

A random variable is a mapping from sample space to a set of real numbers. What does this mean? Let us take the usual evergreen example of "flipping a coin".

In a "coin-flipping" experiment, the outcome is not known prior to the experiment, that is we cannot predict it with certainty (non-deterministic/stochastic). But we know the all possible outcomes – Head or Tail. Assign real numbers to the all possible events (this is called "sample space"), say "0″ to "Head" and "1″ to "Tail", and associate a variable "X" that could take these two values. This variable "X" is called a random variable, since it can randomly take any value '0′ or '1′ before performing the actual experiment.

Obviously, we do not want to wait till the coin-flipping experiment is done. Because the outcome will lose its significance, we want to associate some probability to each of the possible event. In the coin-flipping experiment, all outcomes are equally probable (given that the coin is fair and unbiased). This means that we can say that the probability of getting Head (random variable X = 0) as well that of getting Tail (X =1) is 0.5 (i.e. 50-50 change for getting Head/Tail).

This can be written as,

$$P(X = 0) = 0.5 \; and \; P(X = 1) = 0.5$$

Continuous Vs. Discrete Random Variables:

Random variables can be classified into two types: continuous and discrete. The continuous random variables take on continuous values in a given interval. The discrete random variables take only discrete or separate values. Continuous random variables are characterized by probability density function (PDF) and cumulative distribution function (CDF). The discrete random variables are

characterized by probability mass function (PMF) and cumulative mass function (CMF).

Consider the following experiment, where the angular position of a pointer on a rotating wheel is measured. The rotating wheel is spun with random speed everytime and the angular position of the pointer is read. The following event is defined: Event A is defined as angle of the pointer when it lies in the interval [$\pi/4$, $\pi/2$) radians. Then, we can define a random variable and the probability of the event as follows.

$$Sample\ Space\ S = [0,2\pi)\ radians$$

$$Event\ A: Angle \in [\pi/4,\pi/2)\ radians$$

$$Random\ Variable: X(\lambda) = \lambda\ radians$$

$$P(A) = P(\pi/4 \leq X < \pi/2)$$

Since the above mentioned random variable can take any value (integer or any decimal value) in the given interval, it is an example of continuous random variable.

Consider the following experiment, where two dice are thrown and the numbers on the faces of the two dice are recorded. An event is defined as the occassion where the sum of two numbers shown on the dice is greater than '8'. Then, we can define a random variable and the probability of the event as follows.

$$Sample\ Space\ S = \{(\lambda_1,\lambda_2): \lambda_1,\lambda_2 \in \{'1','2','3','4','5','6'\}$$

$$Event\ A: = \{(\lambda_1,\lambda_2): \lambda_1 + \lambda_2 > 8\}$$

$$Random\ Variable: X(\lambda_1,\lambda_2) = \lambda_1 + \lambda_2$$

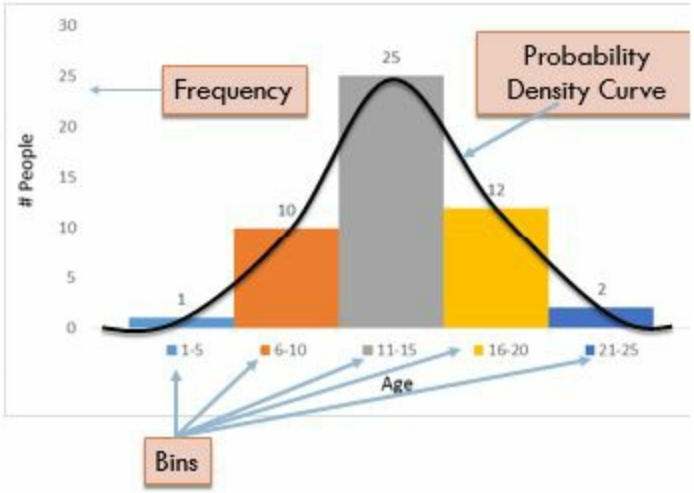
$$P(A) = P(X > 8)$$

Since the above mentioned random variable can take only discrete values (9, 10, 11 or 12) it is an example of discrete random variable.

Histograms and Density functions:

Consider a survey where people are asked to answer "yes" to a particular question. The survey results are categorized according to the age group of the individuals and are tabulated below (on the left). On the right, a bar graph is plotted with age group on the x-axis and the "number of people answered yes to the question" on the y-axis. The categories on the x-axis are called "bins". The values on the y-axis are called "frequencies". Here, the term "frequency" indicates the occurance of some phenomena. This type of bar chart is called "Histogram". It forms the basis for density functions.

Survey	#People who answered "Yes" to a particular question
Age	
1-5	1
6-10	10
11-15	25
16-20	12
21-25	2
Total	50



Now, connect the mid points of the bars in the histogram as shown above. The resultant curve is called probability density curve. Mathematically expressed, it is called probability density function. For a discrete random variable, the curve is characterized by probability mass function (PMF).

Following is a small snippet of Matlab code used to plot the probability density curve for the given histogram for a standard normal distribution. The pdf of the standard normal distribution is given by

$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{\left(-\frac{x^2}{2}\right)}$$

1) Generate random numbers drawn from standard normal distribution (mean=0, variance=1)

```
[f,x]=hist(randn(10000,1),100)
```

Note: Here the variable ‘x’ holds the center of 100 bins and the variable ‘f’ represents the frequency of occurance of the elements in those bins.

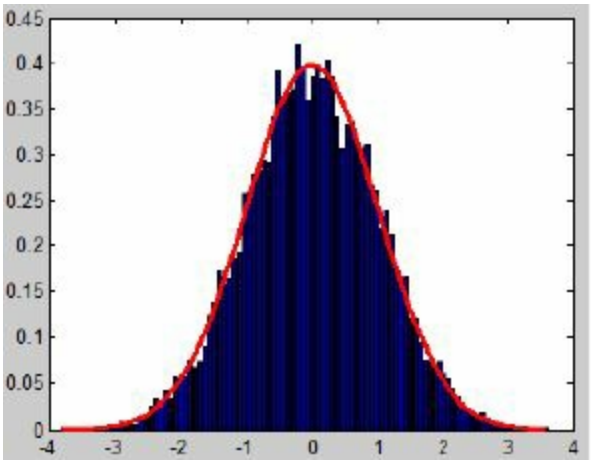
2) Plot it by normalizing with respect to area

```
bar(x,f/trapz(x,f));hold on
```

Note: The ‘trapz’ function in Matlab computes the approximate area under the simulated PDF curve. We know that the area under any PDF curve integrates to unity. Thus we need to normalize the histogram using the total area obtained by the ‘trapz’ function. This brings the total area under the normalized PDF curve to unity. This step is essential, only if we want to match the theoretical and simulated PDF. This method can be applied to any PDF curve.

3) Plot the theoretical probability density curve using the equation for PDF given above

```
g=1/sqrt(2*pi)*exp(-0.5*x.^2);
plot(x,g,’r');hold off
```



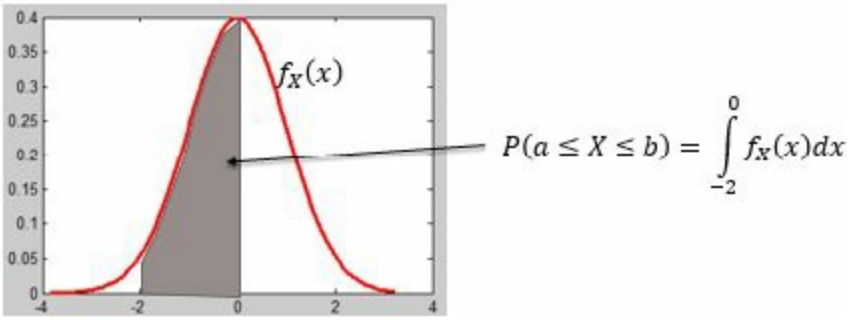
Probability Density Function:

Consider a random variable – denoted by X – that represents the event of drawing a number from a standard normal distribution. This event will have a probability density curve (as shown in the figure above) characterized by a probability density function denoted as $f_X(x)$.

The probability that the random variable X takes on a value in the interval [a, b] is the area under the graph of the density function available within the given interval. It is given by

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx$$

As an example, we can find the probability that the number generated lies in the closed interval [-2, 0] as,



Thus, the probability density function (PDF) records the probabilities associated with the random variable as areas under its graph. For a PDF to be legitimate, it has to satisfy the following conditions (as defined indirectly by the laws of probability)

1) Probability is always positive

$$f_X(x) \geq 0 \; ; \; for\, all\, x$$

2) The maximum value for probability is '1'. That is, the total area under the entire probability density curve is always equal to 1.

$$\int_{-\infty}^{+\infty} f_X(x)dx = 1$$

Probability Density Function to Cumulative Distribution Function:

We know that the Probability Density Function (PDF) gives the probability that a random variable X takes on a value in the interval [a, b] is the area above this interval and under the graph of the density function.

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx$$

Now, what is the probability that the random variable takes on a value up to some number "x"? It is given by

$$P(X \leq x) = \int_{-\infty}^{x} f_X(t)dt$$

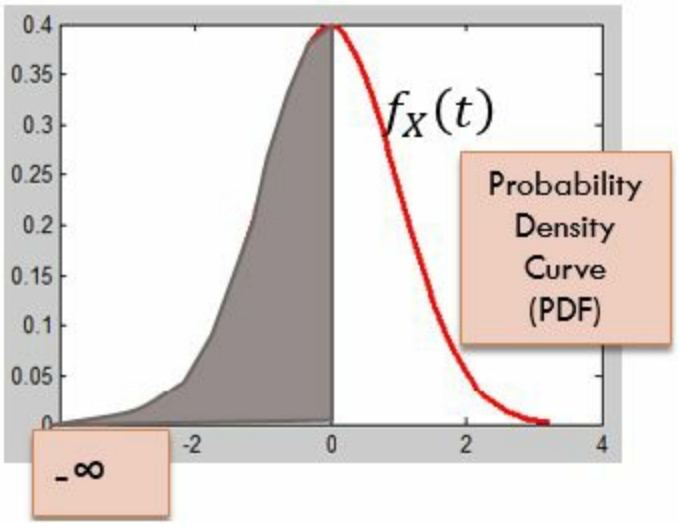
This is called Continuous Distribution Function (CDF) –denoted by $F_X(x)$. Note that the PDF is denoted by "small- f" and the CDF is denoted by 'f' in caps.

Cumulative Distribution Function records the probabilities associated with a random variable X, in a different way. It gives the accumulated probability up to a certain value 'x' (refer the figure below).

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(t)dt$$

Therefore,

$$P(a \leq X \leq b) = \int_a^b f_X(t)dt = \int_{-\infty}^{b} f_X(t)dt - \int_{-\infty}^{a} f_X(t)dt = F_X(b) - F_X(a)$$



The PDF can be converted to CDF and vice versa as

$$F_X(x) = \int_{-\infty}^{x} f_X(t)dt$$

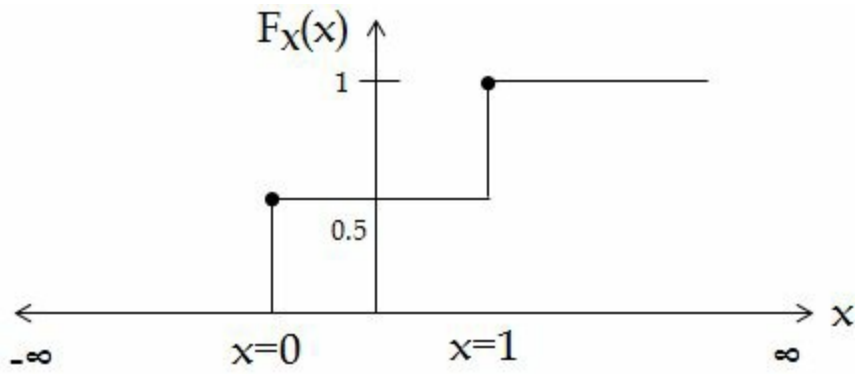
$$f_X(t) = \frac{dF_X(x)}{dx}$$

The above descriptions are for continuous random variables. Mathematically, a complete description of a random variable is given by "Cumulative Distribution Function"- $F_X(x)$. Here the capitalized

letter “X” is a random variable and “x” is a dummy variable which is a place holder for all possible outcomes (“0” and “1” in the above mentioned coin flipping experiment). The Cumulative Distribution Function is defined as,

$$F_X(x) = P(X \leq x)$$

If we plot the CDF for our coin-flipping experiment, it would look like the one shown in the figure below.



The example provided above is of discrete nature, as the values taken by the random variables are discrete (either “0″ or “1″) and therefore the random variable is called Discrete Random Variable. The CDF should be called Cumulative Mass Function (CMF).

If the values taken by the random variables are of continuous nature (Example: Measurement of temperature), then the random variable is called Continuous Random Variable and the corresponding cumulative distribution function will be smoother without discontinuities.

Probability Density function (PDF) and Probability Mass Function (PMF):

It is more common to deal with Probability Density Function (PDF)/Probability Mass Function (PMF) than CDF.

The probability that a continuous random variable X takes on a value in the interval [a, b] is obtained by integrating the PDF over the interval as limits.

$$P(a \leq X \leq b) = \int_a^b f_X(x)dx$$

For a discrete random variable that takes on discrete values, it is common to define Probability Mass Function. The probability that the discrete random variable takes some value X is given directly by the Probability Mass Function.

$$P(X = x) = f_X(x)$$

Note that there is no integration involved in this case. This is an important difference between the way the two parameters, namely, PDF and PMF are calculated.

Based on the probability density function or how the PDF graph looks, PDFs fall into different categories like binomial distribution, Uniform distribution, Gaussian distribution, Chi-square distribution, Rayleigh distribution, Rician distribution etc. Out of these distributions, you will encounter Gaussian distribution or Gaussian Random variable in digital communication very often.

The following section talks about various statistical parameters that can be defined for a random variable.

Mean:

The mean of a random variable is defined as the weighted average of all possible values the random variable can take. Probability of each outcome is used to weigh each value when calculating the mean. Mean is also represented by "expectation operator"- (E[X])

For continuous random variable X and probability density function $f_X(x)$

$$E[X] = \int_{-\infty}^{\infty} x f_X(x) dx$$

For a discrete random variable X, the mean is calculated as weighted average of all possible values (x_i) weighted with individual probability (p_i)

$$E[X] = \mu_X = \sum_{-\infty}^{\infty} x_i p_i$$

Variance:

Variance measures the spread of a distribution. For a continuous random variable X, the variance is defined as

$$var[X] = \int_{-\infty}^{\infty} (x - E[X])^2 f_X(x)\, dx$$

For discrete case, the variance is defined as

$$var[x] = \sigma_X^2 = \sum_{-\infty}^{\infty} (x_i - \mu_X)^2 p_i$$

Standard Deviation (σ) is defined as the square root of variance σ_X^2.

Properties of Mean and Variance:

For a constant – "c" following properties will hold true for mean

$$\begin{aligned} E[cX] &= cE[X] \\ E[X + c] &= E[X] + c \\ E[c] &= x \end{aligned}$$

For a constant – "c" following properties will hold true for variance

$$\begin{aligned} var[cX] &= c^2 var[X] \\ var[X + c] &= var[X] \\ var[c] &= 0 \end{aligned}$$

PDF and CDF define a random variable completely. For example: If two random variables X and Y have the same PDF, then they will have the same CDF and therefore their mean and variance will be the same.

On the other hand, mean and variance describe a random variable only partially. If two random variables X and Y have the same mean and variance, they may or may not have the same PDF or CDF.

******

Back to Table of Contents

4.4 Gaussian random variable and Gaussian distribution

For a Gaussian random variable, the outcomes follow Gaussian distribution. The shape of the Gaussian probability density curve is bell-shaped. It is used most widely in modeling the communication channel. For example, all channels are assumed to be Additive White Gaussian Noise channel. What is the reason behind it? Gaussian noise gives the smallest channel capacity with fixed noise power. This means that it results in the worst channel impairment. So the coding designs done under this most adverse environment will give superior and satisfactory performance in real environments. For more information on "Gaussianity" refer [Pasupathy1989]

The PDF of the Gaussian distribution (also called as Normal Distribution) denoted by $\mathbb{N}(\mu,\sigma^2)$ is completely characterized by its mean (μ) and variance (σ^2),

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{[x-\mu]^2}{2\sigma^2}\right)$$

Since the PDF is defined as the first derivative of CDF, reverse engineering tells us that CDF can be obtained by taking an integral of PDF. Thus to get the CDF of the above given function,

$$F_X(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{x-\mu}{\sigma}} e^{\frac{-t^2}{2}} dt$$

Generating Gaussian Random Numbers in Matlab:

In Matlab "randn" function [MatlabRandn] is used to generate continuous random numbers drawn from standard normal distribution (mean=0, standard deviation=1).

To generate 100 pseudo random numbers drawn from standard normal distribution

```
>> randn(100,1)
```

To generate 100 pseudo random numbers drawn from normal distribution with arbitrary mean 'a' and standard deviation 'b'

```
>> a+b*randn(100,1)
```

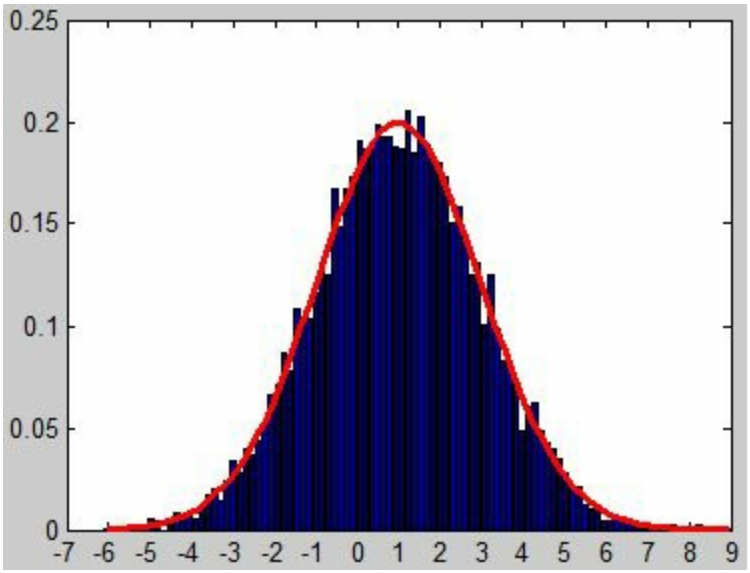
Let's generate 10,000 random numbers drawn from Gaussian distribution (with mean- μ =1, standard deviation σ=2) and test whether the probability density curve follows the characteristic bell shaped curve.

Matlab snippet:

```
>> [f,x]=hist(1+2.*randn(10000,1),100); %Simulated PDF
>> bar(x,f/trapz(x,f));hold on;
>> g=1/sqrt(2*pi*4)*exp(-0.5*(x-1).^2/(4)); %Theoretical PDF
>> plot(x,g,'r');hold off;
```

Note: The 'trapz' function in Matlab computes the approximate area under the simulated PDF curve. We know that the area under any PDF curve integrates to unity. Thus we need to normalize the histogram using the total area obtained by the 'trapz' function. This brings the total area under the normalized PDF curve to unity. This step is essential, only if we want to match the theoretical and simulated PDF. This method can be applied to any PDF curve.

The resultant histogram shows that the generated numbers follow the probability density curve of Gaussian distribution with mean=1 and standard deviation=2.



******

Back to Table of Contents

4.5 Uniform Random Variables and Uniform Distribution

Continuous Uniform Random Variables:

Continuous Uniform Random Variables are used to model a scenario where a continuous random variable can take values that are equally distributed (with equal probability) in an interval. Uniform distribution is the underlying distribution to model such random variables.

A continuous uniform distribution is denoted by

$$X \sim \mathcal{U}(a, b), with\ a \leq b$$

Here "a" and "b", arethe limits of the interval over which the distribution is effectively defined.

Probability Density Function

The Probability Density Function (PDF) of the continuous uniform random distribution defined above

is given by

$$f(x) = \begin{cases} \frac{1}{b-a} & ,if\ a \le x \le b \\ 0 & ,otherwise \end{cases}$$

This equation indicates that the distribution when used, generates values in the interval (a,b) with equal probability (with probability1/[b-a])

Continuous Density Function

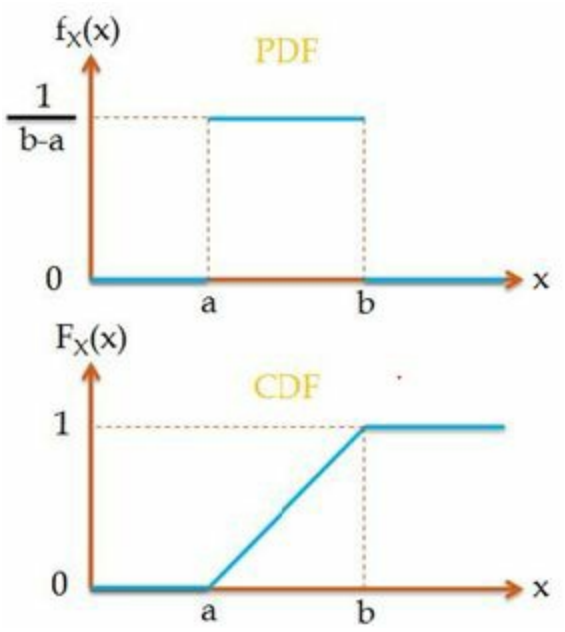
The Continuous Density Function (CDF) can be derived from the above PDF using the relation

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

Therefore, the CDF of the continuous uniform distribution is given by

$$F(x) = \begin{cases} 0 & ,if\ x < a \\ \frac{x-a}{b-a} & ,if\ x \subset [a,b) \\ 1 & ,if\ x \ge b \end{cases}$$

Plotting the PDF and CDF,



Discrete Uniform Random Variable:

Discrete Uniform Random Variables are used to model a scenario where a discrete random variable can take values that are equally distributed (with equal probability) in an interval. The underlying discrete uniform distribution is denoted by

$$X \sim \mathcal{U}(\mathcal{S})$$

Here, $\mathcal{S}$ is a finite set of discrete points on the real line, meaning that all the elements in the finite set are equally probable.

Example:

“Rolling a fair die” can be considered as an experiment to generate uniform integers in the range 1 to

6. Since rolling of a fair die gives only one of the six possible integer values (1, 2, 3, 4, 5, 6), it is an experiment that generates discrete values. Since all the values in a fair die are equally likely (with probability of getting each value =1/6) it can be modeled using a discrete uniform distribution.

For a die, the set S can take values {1, 2, 3, 4, 5, 6}

Probability Mass Function (PMF):

If $S = \{s_1, s_2, \dots, s_n\} \in \mathbb{Z}$, then according to the definition of uniform distribution, each element in the set are equally probable. Therefore,

$$P[X = s_1] = P[x = s_2] = \cdots = P[X = s_n] = \frac{1}{n}$$

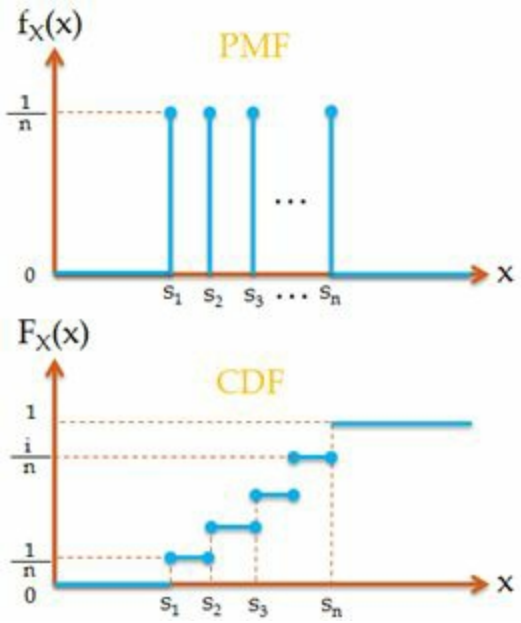
Then the PMF can be given by

$$f(x) = \begin{cases} \frac{1}{n}, x \in \{s_1, s_2, \dots, s_n\} \\ 0, \ otherwise \end{cases}$$

For rolling a fair die, the PMF simplifies as follows

$$f(x) = \begin{cases} \frac{1}{6}, x \in \{1,2,3,4,5,6\} \\ 0, \ otherwise \end{cases}$$

The PMF and CDF for a discrete uniform random distribution is shown below



Generating Uniform Random Numbers in Matlab:

Generating Continuous Uniform Random Numbers:

In Matlab "rand" function [MatlabRand] is used to generate continuous uniform random numbers in the range [0, 1]. The "rand" function is used to pick any number in the interval [0, 1] in which all the numbers in the interval are equally likely.

rand(n,m) will generate a matrix of size nxm. A trial run at my system gave the following results

```
>>rand(3,4)
>>
ans =
    0.2920    0.3900    0.4332    0.4159
    0.0948    0.6802    0.8264    0.7536
    0.6770    0.8449    0.6083    0.7715
```

To generate a random number in the interval [a, b] one can use the following code

a + (b-a)*rand(n,m); %Here nxm is the size of the output matrix

For example, to generate a matrix of size 3×4 using uniform distribution in the interval [2, 10]

```
>> 2 + (10-2)*rand(3,4)
>>
ans=
    5.4852    4.5016    9.6905    2.0585
    2.7066    8.4537    9.8910    4.4816
    3.1029    5.2562    6.9187    8.7710
```

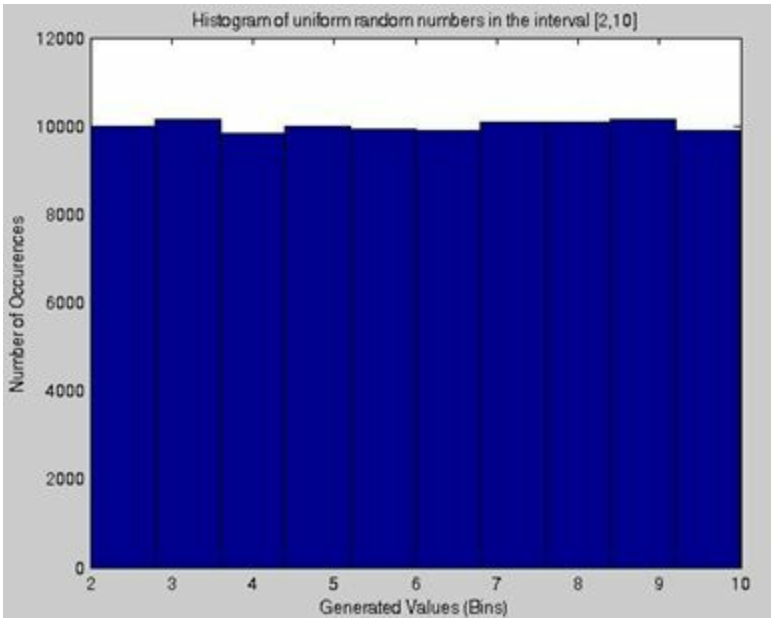
Testing Uniformity:

To test whether the numbers generated by the uniform distribution are uniform in the interval [a, b], one has to generate very large number of values using the rand function and then plot the histogram

Testing the uniformity of rand function in the interval [2, 10]

```
>> x = 2+(10-2)*rand(1,100000);
>> hist(x);
```

The histogram plot given below - shows that the numbers in the range [2, 10] are generated with almost equal probability.



Generating Discrete Uniform Random Numbers:

The same "rand" function can be used to generate discrete random numbers that follow discrete uniform distribution. The code to generate uniformly distributed discrete numbers from the set S= {1,2,3,4,…,n} is given below.

```
r = ceil(n.*rand(1,100));
%This will generate 100 uniformly distributed discrete numbers on the set 1:n
```

To model rolling of a fair die, the sample space is S={1,2,3,4,5,6}. A roll of the die should generate

a number from the sample space with equal probability. This single event can be modeled in Matlab as

```
>> r = ceil(6.*rand(1,1));
```

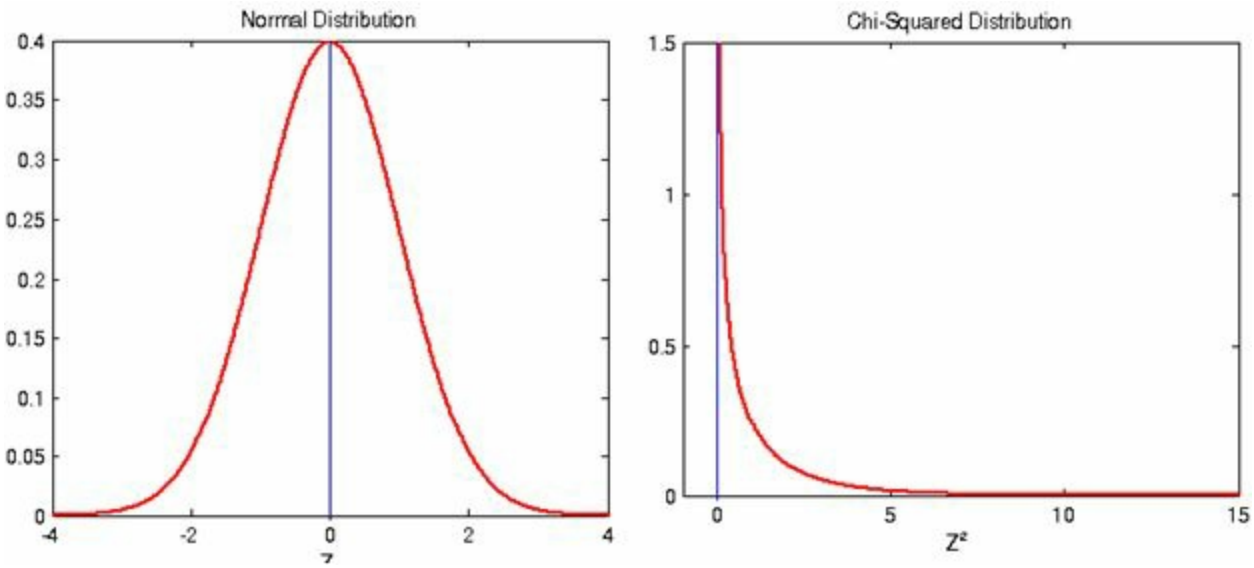
(The uniformity test for discrete random numbers can be done using the same code given above)

******

Back to Table of Contents

4.6 Chi-Squared Random Variable and Chi-Squared Distribution

A random variable is always associated with a probability distribution. When the random variable undergoes mathematical transformation the underlying probability distribution no longer remains the same. Consider a random variable Z whose probability distribution function (PDF) is a standard normal distribution (mean=0 and variance=1). Now, if the random variable is squared (a mathematical transformation), then the PDF of Z^2 is no longer a standard normal distribution. The newly transformed distribution is called Chi-Squared Distribution with 1 degree of freedom. The PDF of Z and Z^2 are plotted in following figure.



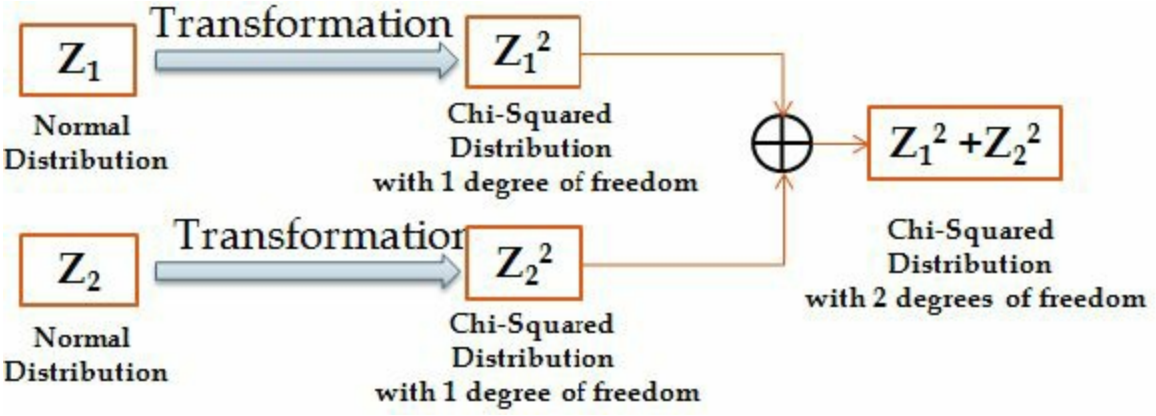
The mean of the random variable Z is E(Z) = 0 and for the transformed variable Z^2, the mean is given by $E(Z^2)=1$. Similarly, the variance of the random variable Z is var(Z)=1, whereas the variance of the transformed random variable Z^2 is $var(Z^2)=2$. In addition to the mean and variance, the shape of the distribution is also changed. The distribution of the transformed variable Z^2 is no longer symmetric. In fact, the distribution is skewed to one side. Also the random variable Z^2 can take only positive values whereas the random variable Z can take negative values too (note the x-axis in both figures above).

Since the new transformation is based on only one parameter (Z), the degree of freedom for this transformation is 1. Therefore, the transformed random variable Z^2 follows - "Chi-squared distribution with 1 degree of freedom".

Suppose, if $Z_1, Z_2, \ldots, Z_k$ are independent random variables that follows standard normal distribution(mean=0,variance= 1), then the transformation,

$$\chi_k^2 = Z_1^2 + Z_2^2 + \cdots + Z_k^2$$

is a Chi-squared distribution with 'k' degrees of freedom. The following figure illustrates the transformation relationship between Chi-Squared distributions with 1 & 2 degrees of freedom and the normal distribution. In the same manner, the transformation can be extended to 'k' degrees of freedom.



The previous equation is derived from k random variables that follow standard normal distribution. For a standard normal distribution, the mean=0. Therefore, the transformation is called "central" Chi-square distribution. If, the underlying 'k' random variables follow normal distribution with non-zero mean, then the transformation is called "non-central" Chi-square distribution. In channel modeling, the central Chi-square distribution is related to Rayleigh Fading scenario and the non-central Chi-square distribution is related to Rician Fading scenario.

Mathematically, the PDF of the central Chi-squared distribution with 'k' degrees of freedom is given by

$$f_{\chi_k^2}(x) = \frac{1}{2^{\left[\frac{k}{2}\right]}\,\Gamma\left(\frac{k}{2}\right)} x^{\left[\frac{k}{2}-1\right]} e^{-\frac{x}{2}}$$

The mean and variance of the central Chi-squared distributed random variable is given by

$$\mu = E(\chi_k^2) = k$$
$$\sigma^2 = var(\chi_k^2) = 2k$$

Applications:

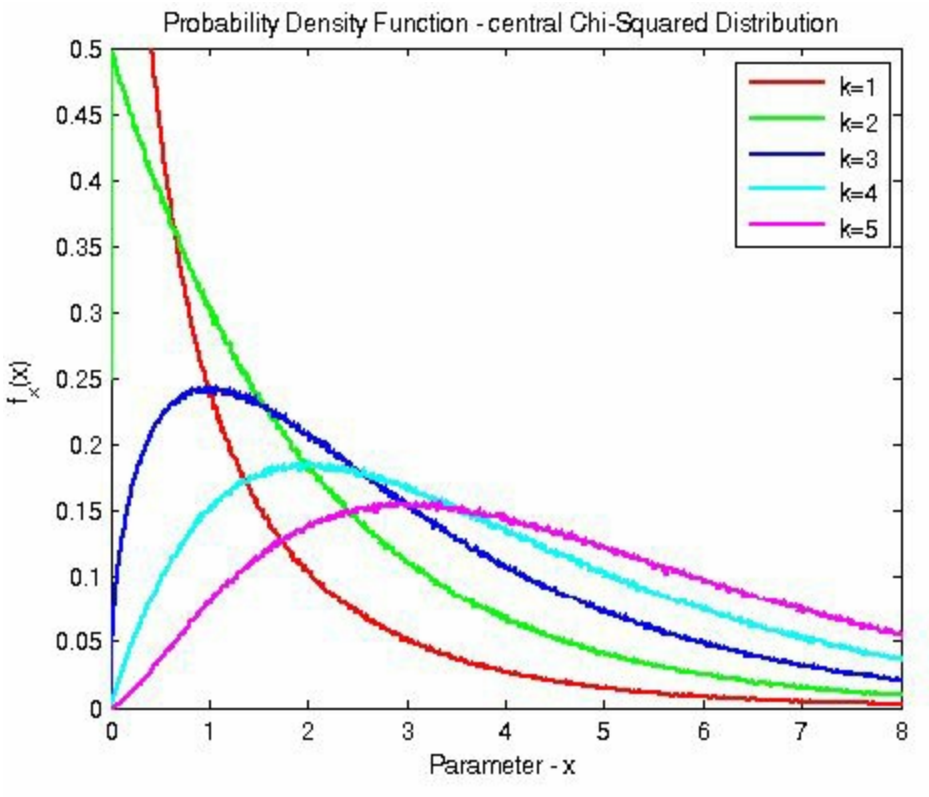
Chi-squared distribution is used in hypothesis testing (to compare the observed data with expected data that follows a specific hypothesis) and in estimating variances of a parameter.

Simulation:

The following Matlab code is used to simulate central Chi-squared distributed variables with degrees of freedom = 1,2,3,4 and 5. Finally the PDFs of the simulated variables are plotted.

```
%Demonstrate PDF of Chi-square Distribution
clear;clc;
k=1:1:5; %degrees of freedom
N = 10000000; %Number of Samples
bins = 0:0.01:10; %bins for plotting histogram
```

```
%allocating space for chi-square RVs and distributions
randNums = zeros(length(k),N);
binPos= zeros(length(k),length(bins));
dist= zeros(length(k),length(bins));
lineColors=['r','g','b','c','m','y','k']; %line color arguments
legendString ={}; %cell array of string for printing legend

%Generating Chi-Square Distribution with k degrees of freedom
index=1;
for i=k, %outer loop - runs for each degrees of freedom
    for j=1:i,
        %Sum of square of k independent standard normal Random Variables
        randNums(index,:) = randNums(index,:) + randn(1,N).^2;
    end

    %Computing Histogram for each degrees of freedom
    [Q,X] = hist(randNums(index,:),bins);
    dist(index,:)  =Q;
    binPos(index,:)=X;

    %plot commands
    plot(binPos(index,:),dist(index,:)/(0.01*N),lineColors(index),'LineWidth',1);
    hold on;
    legendString{index}=strcat(' k=',num2str(index));
    index=index+1;
end
legend(legendString);
title('Probability Density Function - central Chi-Squared Distribution ');
xlabel('Parameter - x');
ylabel('f_X(x)');
axis([0 8 0 0.5]);
```



******

Back to Table of Contents

4.7 Non-central Chi-squared Distribution

If squares of k independent standard normal random variables (mean=0, variance=1) are added, it gives rise to central Chi-squared distribution with 'k' degrees of freedom. Instead, if squares of k independent normal random variables with non-zero mean (mean $\neq$ 0, variance=1) are added, it gives rise to non-central Chi-squared distribution.

The non-central Chi-squared distribution is a generalization of Chi-square distribution. A non-central Chi squared distribution is defined by two parameters: 1) degrees of freedom and 2) non-centrality parameter.

As we know from section, the degrees of freedom specify the number of independent random variables we want to square and sum-up to make the Chi-squared distribution. Non-centrality parameter is the sum of squares of means of the each independent underlying Normal random variable.

The non-centrality parameter is given by

$$\lambda = \sum_{i=1}^{k} \mu_i^2$$

The PDF of the non-central Chi-squared distribution is given by

$$f_{\chi_k}(x;\lambda) = \sum_{n=0}^{\infty} exp\left(-\frac{\lambda}{2}\right)\frac{\left(\frac{\lambda}{2}\right)^n}{n!} f_{\chi_{k+2n}}(x)$$

In the above equation $f_{\chi_k}(x;\lambda)$ indicates the non-central Chi-squared distribution with k degrees of freedom with non-centrality parameter specified by λ and the factor $f_{\chi_{k+2n}}(x)$ indicates the ordinary central Chi-squared distribution with k+2n degrees of freedom.

The factor $e^{-\frac{\lambda}{2}}\frac{\left(\frac{\lambda}{2}\right)^n}{n!}$ gives the probabilities of Poisson distribution. So, the PDF of the non-central Chi-squared distribution can be termed as the weighted sum of Chi-squared probabilities with weights being equal to the probabilities of Poisson distribution.

Method of Generating non-central Chi-squared random variable:

Parameters required: k - the degrees of freedom and λ - non-centrality parameter.

1) For a given degree of freedom (k), let the k normal random variables be $X_1, X_2, ..., X_k$ with variances ${\sigma_1}^2, {\sigma_2}^2, \ldots, {\sigma_k}^2 = 1$ and mean $\mu_1, \mu_2, \ldots \mu_k$.

2) Now, our goal is to add squares of the k independent normal random variables with variances=1 and means satisfying the following criteria

$$\lambda = \sum_{i=1}^{k} \mu_i^2$$

3) Put $\mu_1 = \sqrt{\lambda}$ and $\mu_2, \mu_3, \ldots, \mu_k = 0$.

4) Generate k-1 standard normal random variables with $\mu = 0$ and $\sigma^2 = 1$ and one normal random variable with $\mu = \sqrt{\lambda}$ and $\sigma^2 = 1$.

5) Squaring and summing-up all the k random variables gives the non-central Chi-squared random variable

6) The PDF can be plotted using the histogram method.

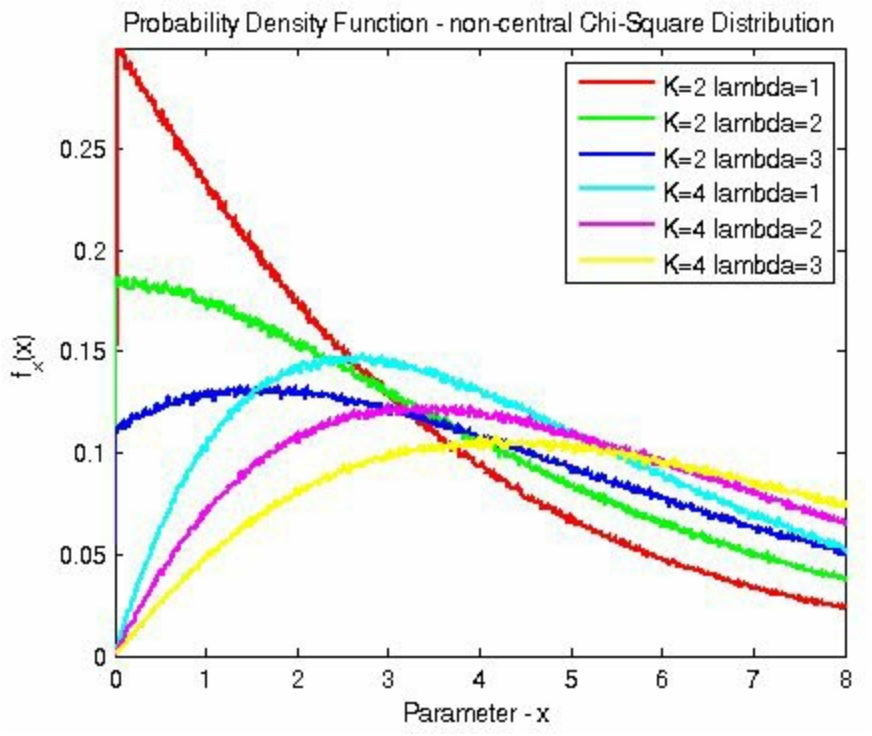
Non-central Chi-squared distribution is related to Rician Distribution and Central Chi-squared distribution is related to Rayleigh distribution.

Matlab Code:

```
%Demonstrate PDF of non-central Chi-squared Distribution
clear;clc;
k=2:2:4; %degrees of freedom
N = 1000000; %Number of Samples
bins = 0:0.01:10; %bins for plotting histogram
lambda = [1,2,3];
%Generating means for all given lambda
means = zeros(length(lambda),max(k))

for i=1:length(lambda)
```

```
    means(i,:) = [sqrt(lambda(i)) zeros(1,max(k)-1)];
end

%allocating space for chi-squared RVs and distributions
randNumsLda1 = zeros(length(k),N);
binPosLda1= zeros(length(k),length(bins));
distLda1= zeros(length(k),length(bins));
randNumsLda2 = zeros(length(k),N);
binPosLda2= zeros(length(k),length(bins));
distLda2= zeros(length(k),length(bins));
randNumsLda3 = zeros(length(k),N);
binPosLda3= zeros(length(k),length(bins));
distLda3= zeros(length(k),length(bins));
lineColors=['r','g','b','c','m','y','k']; %line color arguments
legendString ={}; %cell array of string for printing legend

%Generating Chi-Square Distribution with k degrees of freedom
index=1;
color=1;

for i=k, %outer loop - runs for each degrees of freedom
    for j=1:i,
        %Sum of square of k independent normal Random Variables satisfying non-centrality parameter
        randNumsLda1(index,:) = randNumsLda1(index,:) + (means(1,j)+randn(1,N)).^2;
        randNumsLda2(index,:) = randNumsLda2(index,:) + (means(2,j)+randn(1,N)).^2;
        randNumsLda3(index,:) = randNumsLda3(index,:) + (means(3,j)+randn(1,N)).^2;
    end

    %Computing Histogram for each degrees of freedom
    [Q,X] = hist(randNumsLda1(index,:),bins);
    distLda1(index,:)  =Q;
    binPosLda1(index,:)=X;
    [Q,X] = hist(randNumsLda2(index,:),bins);
    distLda2(index,:)  =Q;
    binPosLda2(index,:)=X;
    [Q,X] = hist(randNumsLda3(index,:),bins);
    distLda3(index,:)  =Q;
    binPosLda3(index,:)=X;

    %plot commands
    plot(binPosLda1(index,:),distLda1(index,:)/(0.01*N),lineColors(color),'LineWidth',1);
    hold on;
    plot(binPosLda2(index,:),distLda2(index,:)/(0.01*N),lineColors(color+1),'LineWidth',1);
    plot(binPosLda3(index,:),distLda3(index,:)/(0.01*N),lineColors(color+2),'LineWidth',1);
    index = index+1;
```

```
    color=color+3;
end
legend('K=2 lambda=1','K=2 lambda=2','K=2 lambda=3','K=4 lambda=1','K=4 lambda=2','K=4
lambda=3');
title('Probability Density Function - non-central Chi-Squared Distribution ');
xlabel('Parameter - x');
ylabel('f_X(x)');
axis([0 8 0 0.3]);
```



A formula for accurately computing values for cumulative non-central chi-square distribution (involving more than 10,000 degrees of freedom) can be found at [Robertson1969]

******

Back to Table of Contents

4.8 Central Limit Theorem

Sometimes I wonder why scientists and engineers in communication industry always plot normal curves; deduce mean, variance, standard deviation etc. to ascertain the implications of any design or development or process improvements or changes. Finally, I discovered that my answer lies in the well-known Central Limit Theorem.

In simpler terms, the theorem states that under certain general conditions, the sum of a large number of independent observations that follow the same distribution has an approximate normal distribution. The approximation steadily improves as the number of observations increase.

The underlying distribution of the independent observation can be anything - binomial, Poisson, exponential, Chi-Square etc. The sum of independent observations following the same underlying

distribution approximates normal distribution.

In other words, CLT states that the sum of independent and identically distributed random variables (with finite mean and variance) approaches Normal distribution as (sample size)$N \to \infty$.

Applications of Central Limit Theorem:

CLT is applied in a vast range of applications including (but not limited to) signal processing, channel modeling, random process, population statistics, engineering research, predicting the confidence intervals, hypothesis testing, even in Casino and gambling (you can find your probability of winning when you gamble at a casino !!!), etc.

One such application in signal processing is - deriving the response of a cascaded series of Low pass filters by applying Central limit theorem. In the article titled "The central limit theorem and low-pass filters" the author has illustrated how the response of a cascaded series of low pass filters approaches Gaussian shape as the number of filters in the series increases [Engelberg2004].

In digital communication, channel noise is often modeled as normally distributed. Modeling a channel as normally distributed when the noise components in that channel are sufficiently large is justified by Central limit theorem.

The following Matlab codes illustrate how the central limit theorem comes to play when the number of observations is increased for two separate experiments: Rolling N unbiased dice and tossing N coins. It is evident from the simulation that as the number of observations (N) is increased; the combined sum of all the N random events approximates Gaussian distribution.

Matlab code – for illustration of CLT using N dice:

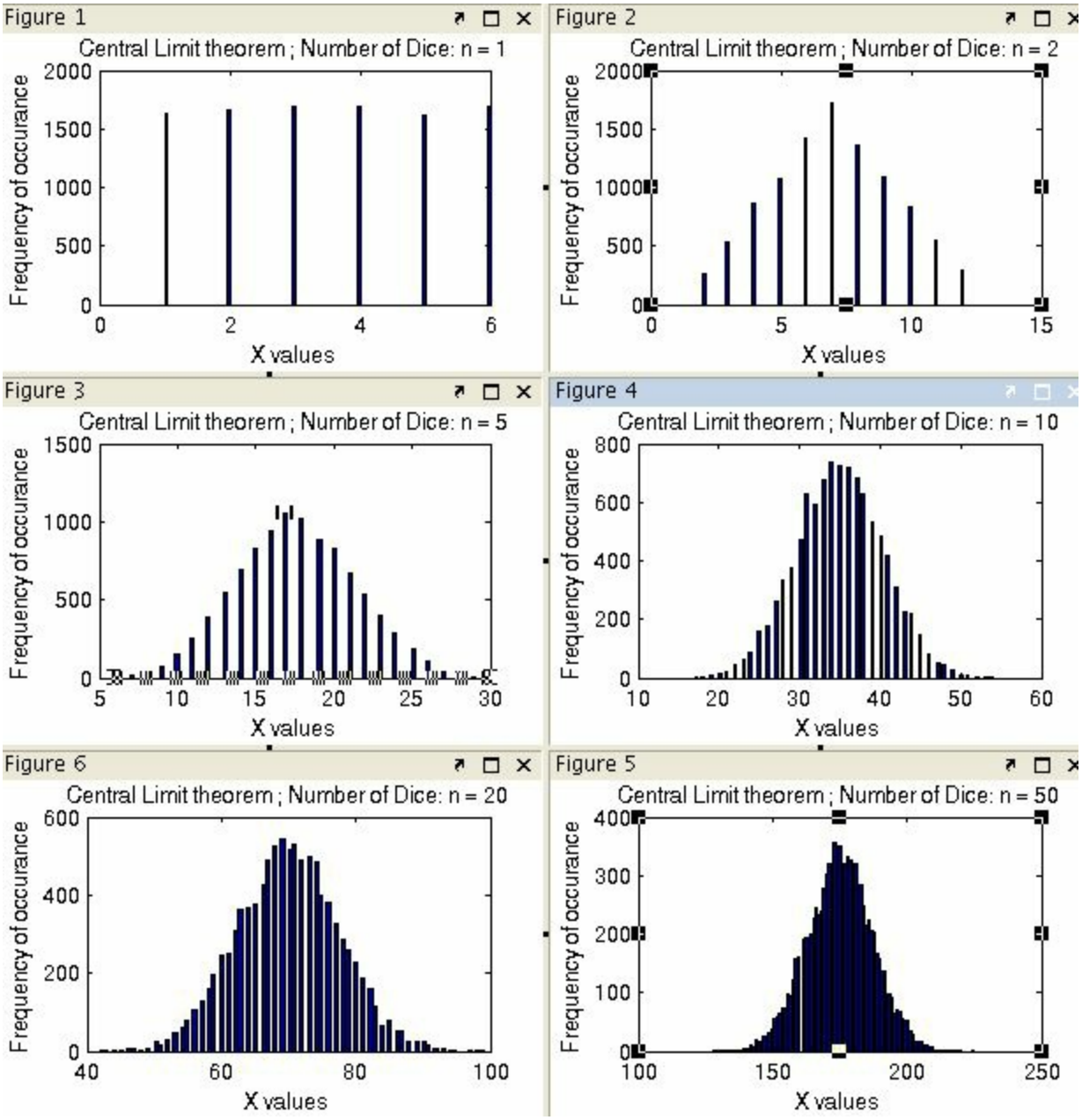
File_1: central_limit_theorem_dice.m:

```
function [output]=central_limit_theorem_dice(n) % Central Limit Theorem for n dice
for i=1:n
    x(i,:)=ceil(6*rand(1,10000)) ;  % Generate 10000 random numbers from 1 to 6 (numbers on a die)
    %independent and identically distributed Random variables
end
for i=1:length(x(1,:)),% Add individual columns to find the sum of n dice.
    y(i)=sum(x(:,i));
end
hist(y,100);% Plot histogram
title(['Central Limit theorem ; Number of Dice: n = ',num2str(n)]);
xlabel('X values');
ylabel('Frequency of occurance');
```

File_2:call_CLT_dice.m

```
number_of_dice = [1 2 3 5 10 15 20 30 50 100];
for i=number_of_dice,
    central_limit_theorem_dice(i);
    pause;
end
```

The simulated output for illustrating the central limit theorem using N dice is given in following figure



Matlab code – for illustration of CLT by flipping N Coins:

File_3: central_limit_theorem_coins.m

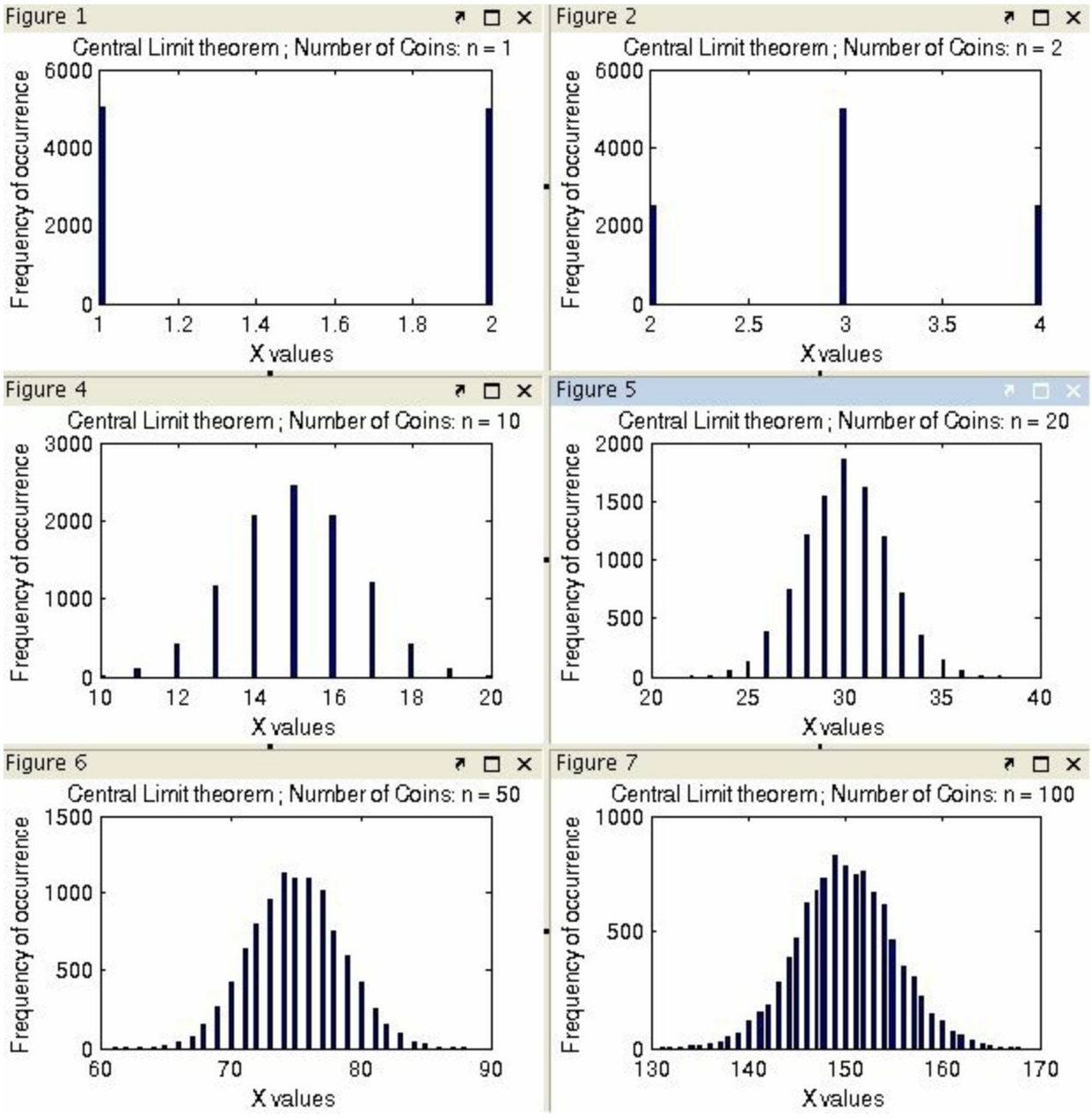
```
function [output]=central_limit_theorem_coins(n) % Central Limit Theorem for n coins

for i=1:n
    x(i,:)=ceil(2*rand(1,10000)) ;% Generate 10000 random numbers 1 and 2 (numbers on a die)
    Head = 1 tail =2
%independent and identically distributed Random variables
end

for i=1:length(x(1,:)), % Add individual columns to find the sum number on n coins.
    y(i)=sum(x(:,i));
end
```

```
hist(y,100);    % Plot histogram
title(['Central Limit theorem ; Number of Coins: n = ',num2str(n)]);
xlabel('X values');
ylabel('Frequency of occurrence');
```

File_4: call_CLT_coins.m

```
number_of_coins = [1 2 3 5 10 15 20 30 50 100];
for i=number_of_coins,
    central_limit_theorem_coins(i);
pause;
end
```

The simulated output for illustrating the central limit theorem using N coins is given in following figure



Note that flipping a coin follows binomial distribution, whereas, as the number of coins increases- the distribution approaches Gaussian/Normal Distribution.

******

Back to Table of Contents

4.9 Colored Noise Generation in Matlab

In communication systems the noise is mostly modeled as white noise. When there exists a noise that is "white", then there must also exist a noise that is colored.

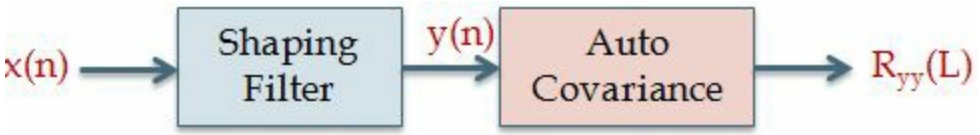
White noise has constant power spectral density across the entire frequency spectrum (theoretically extending up to infinity).There is no correlation between the samples of a white noise process at different time instances i.e. the auto-correlation or the auto-covariance of white noise is zero for all lags except for lag L=0. So the auto-covariance of white noise process will be an impulse function at lag L=0.

Colored Noise:

When the power spectral density of the noise is not uniform across the entire frequency spectrum, it is called colored noise. For a comprehensive definition of various types of colored noise - refer [WikiColorNoise]. There exist non-zero values for auto correlation or auto covariance at different time instances for the colored noise. The auto covariance is maximum for zero lag (L=0) and generally decreases gradually for increasing and decreasing values of lag (L).

Generation of Colored Noise:

Colored noise can be generated by passing the white noise through a shaping filter. The shaping filter is a dynamic filter, usually a low pass filter. The response of the colored noise can be varied by adjusting the parameters of the shaping filter. The low pass filter can be implemented in various ways in Matlab. The Matlab "filter" function is used in this simulation.



$$y(n) = a*y(n-1) + (1-a)x(n)$$

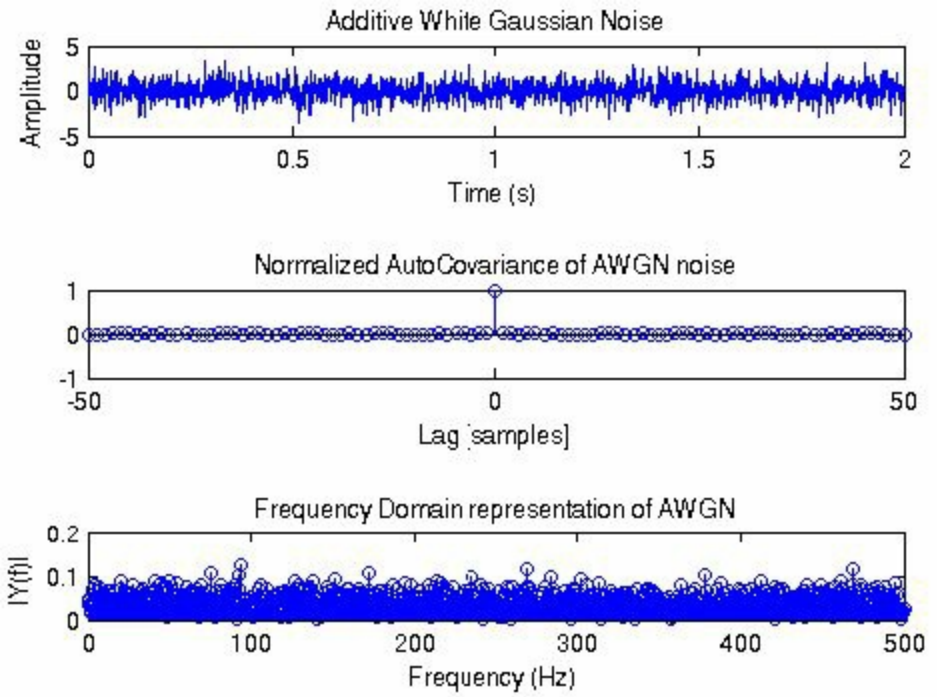
Matlab Code:

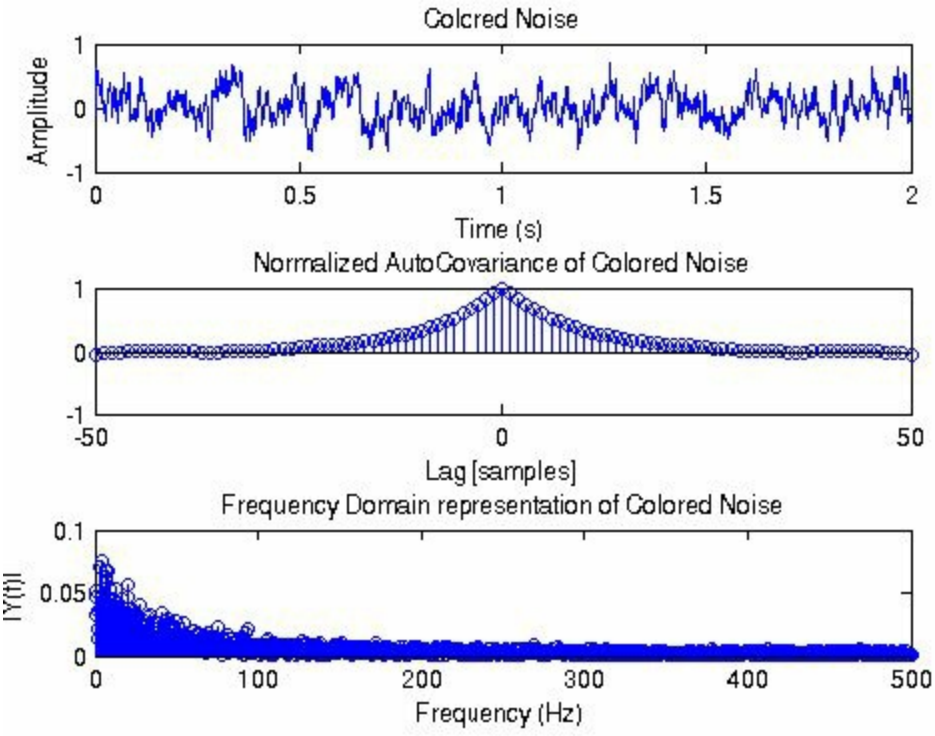
```
%Generation of colored noise in Matlab
a=0.9; %low pass filter parameter
L=50 ; % Number of samples used in auto-covariance calculation
clc;
Fs=1000; %sampling rate
Fc=10; % carrier frequency for the dummy signal
t=0:1/Fs:2; %time base
variance = 1; %variance of white noise

%generate a dummy signal - Later on this can be replaced by the signal that you are interested in , so that the generated noise can be added to it.
```

```
signal=5*sin(2*pi*Fc*t);

% Generate Gaussian White Noise with zero mean and unit variance
whiteNoise=sqrt(variance)*randn(1,length(signal));

%Calculate auto Covariance of the generated white noise  L is the number of samples used in
autocovariance calculation
[whiteNoiseCov,lags] = xcov(whiteNoise,L);

%Frequency domain representation of noise
NFFT = 2^nextpow2(length(noise));
whiteNoiseSpectrum = fft(noise,NFFT)/length(noise);
f = Fs/2*linspace(0,1,NFFT/2+1);

%Colored Noise Generation
x=whiteNoise;

%First Order Low pass filter y(n)=a*y(n-1)+(1-a)*x(n)
%Filter Trasfer function Y(Z) = X(Z)*(1-a)/(1-aZ^-1)
[y zf]=filter(1-a,[1 -a],x);
coloredNoise = y;
[coloredNoiseCov,lags] = xcov(coloredNoise,L);
NFFT = 2^nextpow2(length(coloredNoise));
coloredNoiseSpectrum = fft(coloredNoise,NFFT)/length(coloredNoise);
f = Fs/2*linspace(0,1,NFFT/2+1);

%plotting commands
figure(1);
subplot(3,1,1);
plot(t,whiteNoise);
title('Additive White Gaussian Noise');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,2);
stem(lags,whiteNoiseCov/max(whiteNoiseCov));
title('Normalized AutoCovariance of AWGN noise');
xlabel('Lag [samples]');

subplot(3,1,3);
stem(f,2*abs(whiteNoiseSpectrum(1:NFFT/2+1)))
title('Frequency Domain representation of AWGN');
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')

figure(2);
```

```
subplot(3,1,1);
plot(t,coloredNoise);
title('Colored Noise');
xlabel('Time (s)');
ylabel('Amplitude');
subplot(3,1,2);
stem(lags,coloredNoiseCov/max(coloredNoiseCov));
title('Normalized AutoCovariance of Colored Noise');
xlabel('Lag [samples]');

subplot(3,1,3);
stem(f,2*abs(coloredNoiseSpectrum(1:NFFT/2+1)))
title('Frequency Domain representation of Colored Noise');
xlabel('Frequency (Hz)')
ylabel('|Y(f)|')
```

Simulation output:





******

Back to Table of Contents

Chapter 5: Channel Models and Fading

5.1 Introduction to Channel models

The medium between the transmitting antenna and the receiving antenna is generally termed as "channel". In wireless transmission, the characteristics of the signal change as it travels from the transmitter to the receiver. The signal characteristics are due to several phenomena: 1) existence of line of sight path between the antennas, 2) reflection, refraction and diffraction of the signal due to the objects in between the antennas, 3) The relative motion between the transmitter and receiver and the objects in between them, 4) The signal attenuation as it travels through the medium, 5) Noise The received signal can be obtained from the transmitter signal if we can accurately model the channel in between the antennas. It is quite difficult to model the real world environment. Scientists and engineers have studied various environments and provide us with a way to model the various medium that approximate the real world environments.

In general, the transmitted waveforms across a channel, may experience effects like

1. Reflection
2. Absorption
3. Attenuation (scaling of amplitude)
4. Dispersion (spreading) in time
5. Diffraction (bending of waveform around the edges of obstacles whose size is comparable to the transmitted wavelength)
6. Refraction (bending due to change in refractive indices of two or more mediums)

Thus the received signal can be described by two phenomena - 1) modification of the transmitted signal (characterized by the channel impulse response) and 2) addition of noise. Mathematically it is expressed in time domain as

$$y(n) = h(n) * s(n) + w(n)$$

Or in Matrix form,

$$y = Hs + w$$

Here, y(n) is the received signal, h(n) is the channel impulse response, s(n) is the transmitted signal and w(n) is the additive noise

Depending on the channel properties, a channel (channel impulse response) can be

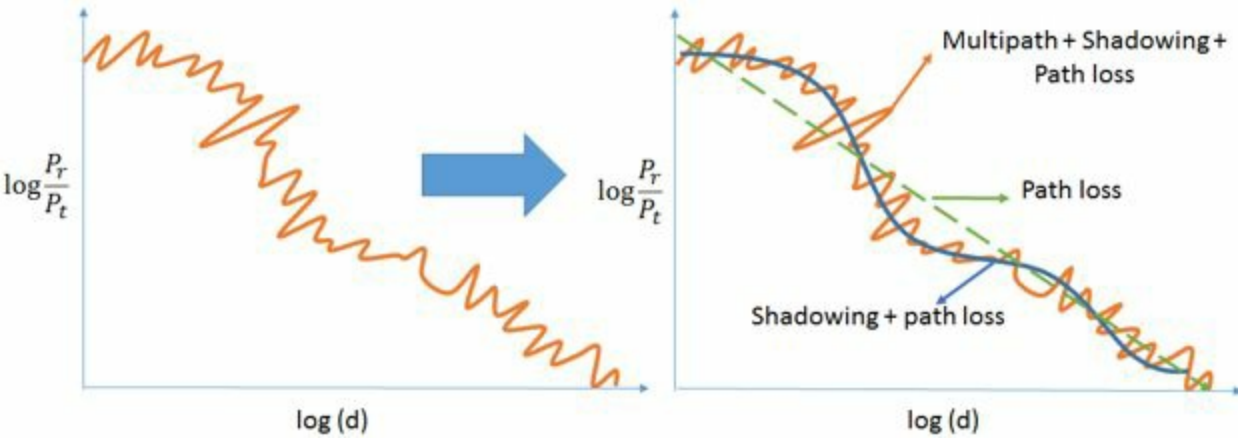
1. Linear/non-linear
2. Time-variant/time-invariant
3. Frequency flat/frequency selective (time dispersive)

The additive noise can be

1. Gaussian/Non-Gaussian
2. Noise correlated in time/frequency
3. Noise correlated spatially (like in MIMO systems)
4. Noise correlated across multiple users

Characterizing Amplitude fluctuations:

Consider a transmitter (Base station) and receiver (mobile) in a city environment. The medium between the transmitter and receiver are obstructed by several buildings, trees and other objects. The receiver keeps moving away from the transmitter. We measure the signal power at the receiver as we move away from the receiver. The ratio of received signal power to the transmitted signal power is plotted against the distance and we obtain a graph as shown in the left hand side plot below.



At first the signal appears very random. Upon closer look we can break it down into three main components as shown in the right half of the plot [Goldsmith2005].

1) Propagation Path Loss
2) Shadowing or large scale fading
3) Multipath fading or small scale fading

Propagation Path Loss:

Path loss modelling is used to model following scenarios:

* Predict the signal loss between the transmitter and the receiver when a clear unobstructed path exists between them.

*Used to model the attenuation effect by various environments as the receiver and transmitter are separated by a distance.

The clear unobstructed path between the transmitter and the receiver is called Line Of Sight (LOS) path.

The three mostly encountered modelling methods available for path loss modelling are

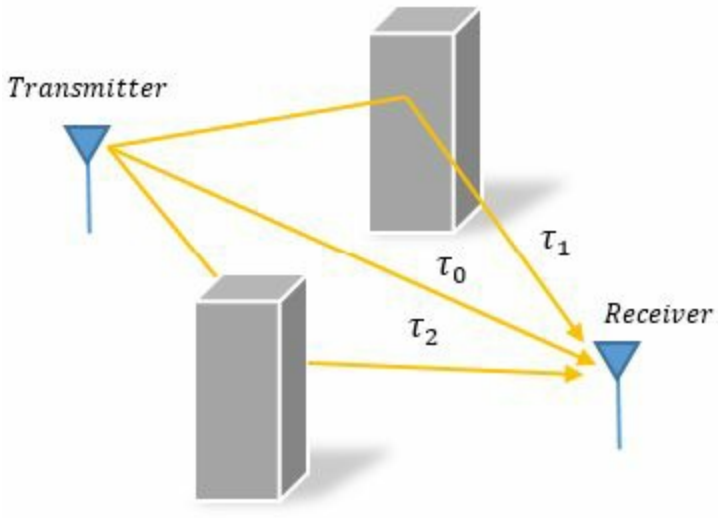
* Friis Free Space Propagation Model – models path loss
* Log Distance Path Loss or Log Normal Shadowing Model – models both path loss and shadowing
* Empirical models:
    * Hata - Okumura Model
    * COST 231 Extension to Hata Model
    * COST 231-Walfish-Ikegami Model
    * Erceg Model
    * Stanford University Interim (SUI) Channel Models
    * ITU Path Loss Models

Shadowing:

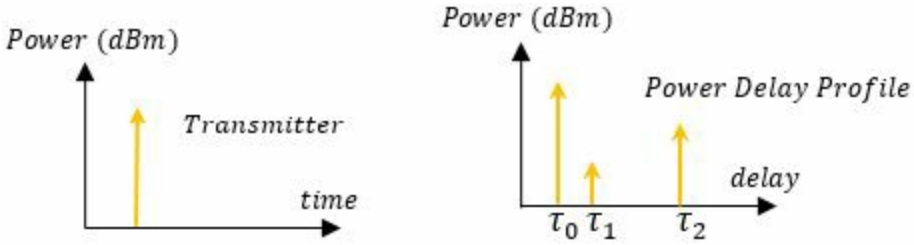
If the environment contains objects like buildings and trees, some part of the transmitted signal gets affected by absorption, reflection, diffraction and scattering. This effect is called shadowing. It is also referred as long-term or large scale fading.

Multipath fading:

A signal travelling in an environment may get reflected by several objects on the path. This gives rise to several reflected signals. The reflected signals arrive at the receiver at different time instants and with different intensities leading to multipath propagation. Depending on the phase of each individual reflected signal, the received signal power may increase or decrease due to contructive or destructive interference. A small variation in the phase of each reflected signal from each multipath may lead to significant difference in the total received power. This phenomenon is also referred as short-term or small scale fading.



The multiple reflected copies of the transmitted signal arrive at the receiver at different time instants and at different power levels. This characteristic of the multipath phenomena is described by Power Delay Profile (PDP) as shown below. Power Delay profile gives a list of different time delays of each multipath (time delays are due to different length of the each individual reflected path) and the associated average power of each path.



The maximum delay for which the received power level becomes negligible is called maximum delay spread τ_{max}. Sometimes Root-Mean-Square value τ_{rms} is used instead of the maximum value. The above shown figure is an example for discrete-time model, where the PDP takes discrete values at discrete time instants. A continuous-time model is also available, where delay profile is specified for continuous time instances.

If too many multiple paths exist, describing path attenuations for each path becomes complicated. Therefore, the path attenuations are modeled as random processes in most of such complicated cases.

******

Back to Table of Contents

5.2 Friis Free Space Propagation Model

Friis Free Space Propagation Model:

It is used to predict the path loss when there is a clear unobstructed LOS between the transmitter and the receiver. It is based on the inverse square law of distance which states that the received power (P_r) decays by a factor of square of the distance (d) from the transmitter.

$$P_r \propto \frac{1}{d^2}$$

The receiver power is obtained by the following equation

$$P_r(d) = P_t \frac{G_t G_r \lambda^2}{(4\pi d)^2 L}$$

$P_r(d)$=Received signal power in Watts expressed as a function of separation –d meters between the transmitter & receiver

P_t= Power at which the signal was transmitted in Watts

G_t, G_r = Gains of transmitter & receiver antennas respectively

λ = Wavelength of transmission in meters.

L = Other Losses that is not associated with propagation loss. It includes system losses like loss at the antenna, transmission line attenuation, loss at various filters etc. The factor is usually greater than or equal to 1. If there are no such system losses L=1.

Friis Free space equation is valid only in the far field region of the transmitting antenna. Refer [AgilentNearFar] to know more about the differences between near field and far field of an antenna.

The propagation path loss-P_L is the loss incurred by the signal during propagation. It is the difference between the effective transmitted power $(P_t G_t)_{dB}$ and the received power at the face of the antenna. Since it is the propagation loss, it does not take into account the gain of the receiver antenna and the other system losses (L). Neglecting the gain of the receiver antenna, the propagation loss is given by

$$P_L\, dB = -10\, log_{10}\left(\frac{\lambda^2}{(4\pi d)^2}\right) = +\, 20\, log_{10}\left(\frac{4\pi d}{\lambda}\right)$$

The above equation implies that the received power decreases at the rate of 20 dB/decade – i.e. If we measure the signal power at a distance d_0 , then the signal power measured at a distance $10*d_0$ will be 20 dB lesser. See the simulation plot below for more details.

We know that the wavelength of an electromagnetic wave is related to its speed (c) and it frequency as

$$\lambda = \frac{c}{f}$$

Where, c is the velocity of the electromagnetic wave = $3*10^8$ m/s and f is the frequency of transmission in Hertz. Thus the Friis Free Space equation can also be written as

$$P_r(d) = P_t \frac{G_t G_r c^2}{(4\pi d f)^2 L}$$

This equation shows that the signal attenuates more for higher frequency. As the frequency increases the path loss also increases. This is the primary reason for why the mobile phone systems are not operated at more than 2GHz frequency. Even though there is enough space beyond this frequency limit, the losses will be more and it will drain the battery quickly.

The gain (G) of the antenna is related to its effective aperture (A_e), which in-turn depends on the physical aperture of the antenna.

$$G = \frac{4\pi A_e}{\lambda^2}$$

Thus the Friis equation can also be written in terms of effective apertures of transmitter (A_{te}) & receiver (A_{re}) as

$$P_r(d) = P_t \frac{A_{te} A_{re}}{d^2 \lambda^2 L}$$

Usually the transmitter power and the receiver power are specified in terms of dBm (Power in decibels with respect to 1mW) and the antenna gains in dBi (Gain in decibels with respect to an isotropic antenna). Therefore, it is often convenient to work in log domain instead of linear domain. Another alternative form of Friis Free space equation in log domain is given by

$$P_r(d)\ dBm = P_t dBm + 10\log_{10}(G_t) + 10\log_{10}(G_r) + 20\log_{10}(\lambda) - 20\log_{10}(4\pi d) - 10\log_{10}(L)$$

Or simply,

$$P_r(d)\ dBm = P_t dBm + G_t\ dBi + G_r\ dBi + 20\log_{10}(\lambda) - 20\log_{10}(4\pi d) - 10\log_{10}(L)$$

Example:

Consider a ground station antenna of gain 25 dBi transmits a signal at 52 dBm. A satellite located at a distance of 41935Km receives the signal. The gain of the receiver antenna is 15 dBi. The frequency of transmission is 1GHz and there is no system loss (L=1). Calculate the received power in dBm at the satellite receiver.

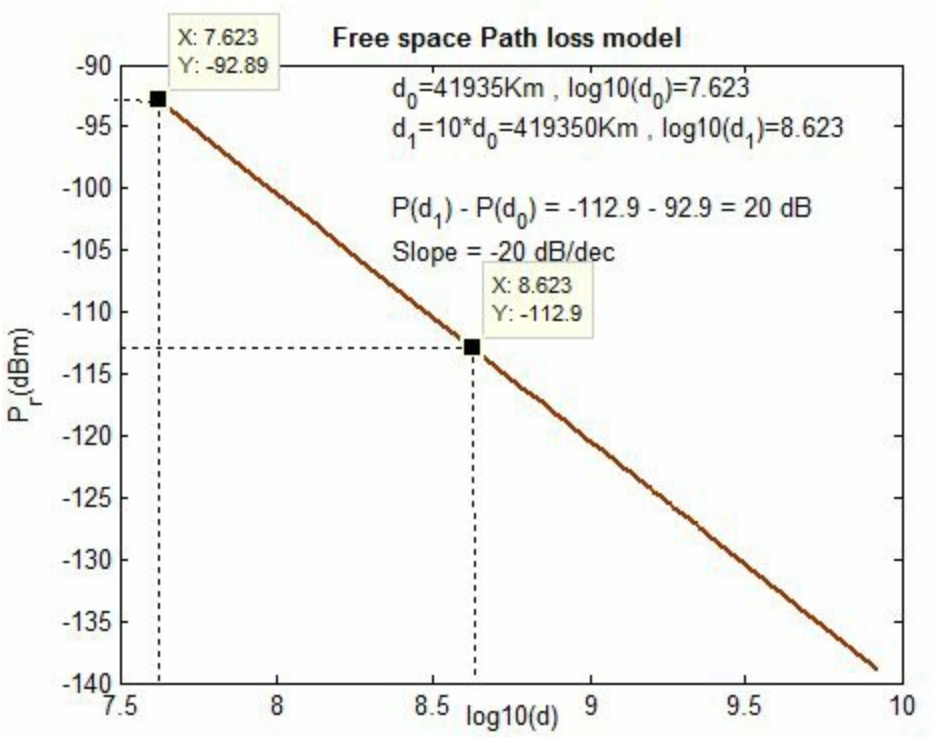
$$P_r(d)\ dBm = 52 + 2 + 15 + 20\log_{10}\left(\frac{3 \times 10^8}{1 \times 10^9}\right) - 20\log_{10}(4\pi \times 41935 \times 10^3)$$
$$= -92.89\ dBm$$

Matlab code to simulate Friis Free Space Equation:

```
%Matlab code to simulate Friis Free space equation

%----------Input section----------------------
PtdBm=52; %Input – Transmitted power in dBm
GtdBi=25; %Gain of the Transmitted antenna in dBi
GrdBi=15; %Gain of the Receiver antenna in dBi
frequency=1*10^9; %Transmitted signal frequency in Hertz
%Example Frequency = 1 GHz
d =41935000*(1:1:200) ; %Array of input distances in meters
L=1; %Other System Losses, No Loss case L=1
%-------------------------------------------------
%Convert all powers to linear scale
Pt=10^((PtdBm-30)/10); %Convert to Watts
Gt=10^(GtdBi/10);
Gr=10^(GrdBi/10);
lambda=3*10^8/frequency; %Wavelength in meters
```

```
Pr= Pt*(Gt*Gr*lambda^2)./((4*pi.*d).^2*L);
PrdBm=10*log10(Pr)+30; %Convert to dBm
plot(log10(d),PrdBm);
title('Free space Path loss model')
xlabel('log10(d)');
ylabel('P_r (dBm)');
```

Simulation Results:



******

Back to Table of Contents

5.3 Log Distance Path Loss or Log Normal Shadowing Model

Log distance path loss model is a generic model and an extension to Friis Free space model. It is used to predict the propagation loss for a wide range of environments, whereas, the Friis Free space model is restricted to unobstructed clear path between the transmitter & the receiver.

In the far field region of the transmitter ($d \geq d_f$), if $PL(d_0)$ is the path loss measured in dB at a distance d_0 from the transmitter, then the path loss (the loss in signal power measure in dB when moving from

distance d_0 to d) at an arbitrary distance d >d_0 is given by

$$PL_{d_0 \to d}(dB) = PL(d_0) + 10n \log_{10}\left(\frac{d}{d_0}\right) + \chi \qquad d_f \le d_0 \le d$$

$PL(d_0)$ = Path Loss in dB at a distance d_0

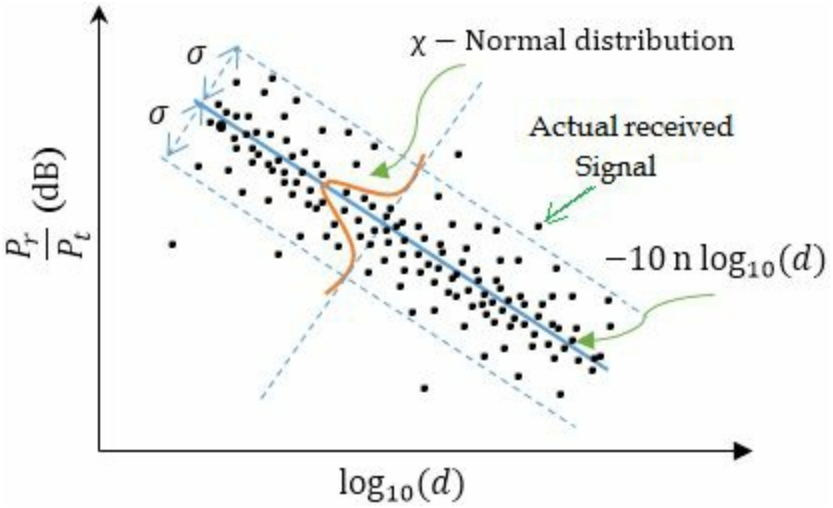
$PL_{d->d0}$ = Path Loss in dB at an arbitrary distance d

n = Path Loss exponent. See the table below that gives the path loss exponent for various environments.

χ = A zero-mean Gaussian distributed random variable (in dB) with standard deviation – σ. This variable is used only when there is a shadowing effect. If there is no shadowing effect, then this variable is zero. Taking log of the Normal (Gaussian)-variable results in the name "Log-Normal" fading.

Environment	Path Loss Exponent (n)
Free Space	2
Urban area cellular radio	2.7 to 3.5
Shadowed urban cellular radio	3 to 5
Inside a building – Line of Sight	1.6 to 1.8
Obstructed in building	4 to 6
Obstructed in Factory	2 to 3

Usually to model real environments the shadowing effects cannot be neglected. If the shadowing effect is neglected, the Path Loss is simply a straight line (see the plot below). To add shadowing effect a zero-mean Gaussian random variable with standard deviation – σ is added to the equation. The actual path loss may still vary due to other factors. Thus the path loss exponent (modeling the slope) and the standard deviation of the random variable should be known precisely for a better modeling.

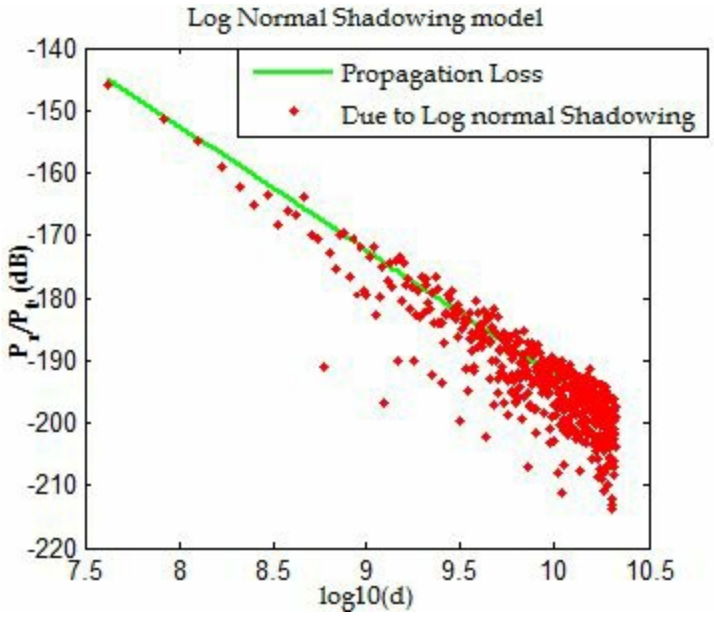


The Path Loss Exponent (PLE) table given above is for reference only. It may or may not fit the actual environment we are trying to model. PLE is an important parameter and it affects the system performance drastically. Usually PLE is considered to be known a-priori but mostly that is not the case. Care must be taken to estimate the PLE for the given environment before design & modeling. PLE estimation is done by equating the observed (empirical) values over several time instants to the established theoretical values. Refer [Srinivasa2009] for a literature on PLE estimation in large wireless networks.

Matlab Code:

```
%Matlab code to simulate Log distance/Log Normal Shadowing Path Loss model

%----------Input section----------------------
PtdBm=52; %Input – Transmitted power in dBm
GtdBi=25; %Gain of the Transmitted antenna in dBi
GrdBi=15; %Gain of the Receiver antenna in dBi
frequency=1*10^9; %Transmitted signal frequency in Hertz
%Example Frequency = 1 GHz
d =41935000*(1:1:500) ; %Array of input distances in meters
L=1; %Other System Losses, No Loss case L=1
sigma = 1; %Standard deviation of zero-mean Normal distribution

%-------------------------------------------------
%Convert all powers to linear scale
Pt=10^((PtdBm-30)/10); %Convert to Watts
Gt=10^(GtdBi/10);
Gr=10^(GrdBi/10);
lambda=3*10^8/frequency; %Wavelength in meters
Pr= Pt*(Gt*Gr*lambda^2)./((4*pi.*d).^2*L);
X = sigma*randn(size(Pr));
propLoss = Pr./Pt;
PLdBm = 10*log10(propLoss)+10*log10(X);
PrdBm=10*log10(Pr)+30; %Convert to dBm
plot(log10(d),10*log10(propLoss),'G','LineWidth',2);
title('Log Normal Shadowing model')
xlabel('log10(d)');
ylabel('P_r/P_t (dB)');
hold on;
plot(log10(d),PLdBm,'r.');
legend('Propagation Loss','Due to Log normal Shadowing');
```

Simulation Results:



******

Back to Table of Contents

5.4 Hata – Okumura Models

In 1986, Yoshihisa Okumura travelled around Tokyo city and made measurements for the signal attenuation from base station to mobile station. He came up with a set of curves which gave the median attenuation relative to free space path loss. Okumura came up with three set of data for three scenarios: 1) Open area 2) Urban area 3) Sub-urban area.

Since this is one of the very first model developed for cellular propagation environment there exist other difficulties and concerns related to the applicability of the model.

If you want to use Okumura model in your computer simulations you have to digitize those curves provided by Okumura and use them in the form of Look Up Tables (LUT). Since it is based on empirical studies, the validity of parameters is limited in range. The parameter values outside the range can be obtained by extrapolation of the curves. But the reasonability of the extrapolation depends on particular given environment. There are also concerns related to the calculation of effective antenna height. Thus every RF modeling tool incorporates its own interpretations and adjustments when it comes to implementing Okumura model.

Hata, in 1980, came up with closed form expressions based on curve fitting of Okumura models. It is most referred macroscopic propagation model. He extended the Okumura models to include effects due to diffraction, reflection and scattering of signal by the surrounding structures in a city environment.

The received power Level in dBm is given by

$$P_r\ dBm = P_t\ dBm + G_t\ dBi - PL\ dB$$

The generic closed form expression for Path Loss is given by

$$PL\ (dB) = A + B\log_{10}(d) + C$$

Here A, B and C are variables that depend on the frequency of transmission (f_c)and antenna heights and the Tx-Rx separation distance ‘d’ is specified in Kilometers. The factor A & B is given by

$$A = 69.55 + 26.16\log_{10} f_c - 13.82\ \log_{10} h_b - a(h_m)$$

$$B = 44.9 - 6.55\ \log_{10}(h_b)$$

f_c = Frequency of transmission in MHz – Valid range – 150MHz – 1500 MHz
h_b= Effective height of tranmitting base station antenna in meters – Valid range 30m – 200m
h_m=Receiving mobile device antenna height in meters – Valid range 1m – 10m

The function a(h_{re}) and the factor C depend on the environment:

For small & medium sized cities:

$$a(h_m) = (1.1\ \log_{10}(f_c) - 0.7)h_m - (1.56\ \log_{10}(f_c) - 0.8)\ \ dB$$
$$C - 0$$

For large cities:

$$a(h_{re}) = \begin{cases} 8.29(\log_{10}(1.54h_{re}))^2 - 1.1\ dB & 150MHz \leq f_c \leq 200MHz \\ 3.2(\log_{10}(11.75h_{re}))^2 - 4.97\ dB & 200MHz \leq f_c \leq 1500MHz \end{cases}$$
$$C = 0$$

For suburban areas:

$$C = -2\left[\log_{10}\left(\frac{f_c}{28}\right)\right]^2 - 5.4$$

For open rural areas:

$$C = -4.78[\log_{10}(f_c)]^2 + 18.33\log_{10}(f_c) - 40.98$$

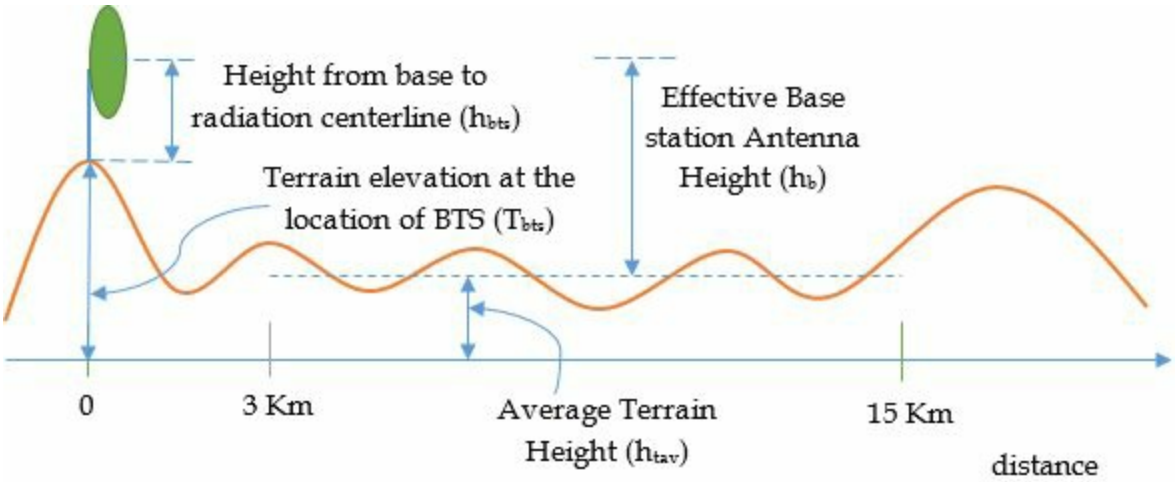
For suburban and rural areas, the factor $a(h_{re})$ is same as that of small & medium cities.

Note: Since Okumura made the measurements in Japan, the model is somewhat specific to Japanese environments.The terms "urban", "sub-urban" and "rural" were not clearly defined and it may lead to different interpretations by people from different backgrounds. The factor "C" is called "area-adjustment factor". It has to be obtained from the measurement data for the actual environment that you are trying to model.

Calculation of Effective base station antenna height (h_b):

It should be noted that the factors A and B are calculated using the effective antenna height of the base station antenna. The effective antenna height is defined as the height of the antenna radiation above the average terrain. Consider a terrain as shown below. First the terrain heights at various points are measured from 3Km to 15km distance from the base station antenna along the direction of propagation.

The values are averaged to give the average terrain height value. The effective antenna height (h_b) is determined as the difference between the height of the antenna (from sea level) and the averaged terrain height.



The effective antenna height for the base station antenna (h_b) is calculated as

$$h_b = h_{bts} + T_{bts} - h_{tav}$$

Where, h_{bts}=Height measured from the base of the BTS tower to the radiation centerline in meters

T_{bts} = Terrain elevation at the location of the BTS in meters
H_{tav}=Height of the average terrain (from 3 Km to 15 km distance from the BTS) in meters

Example:

Lets use Hata model for a medium city to find the received signal levels at various distances = 3, 6, 9, 12 and 15 Kilometers from the base station operating at 870 MHz. The following data is given:
Height of the radiation centerline measured from the base of the BTS = 50 m
Terrain elevation at the location of the BTS = 350 m
Average terrain height = 300 m
Height of the mobile antenna = 3 m
Power delivered to the BTS antenna = 19.5 W
Gain of the BTS antenna = 10 dBi
Gain of the mobile antenna = 0 dBi

Running the following simulation code with the above mentioned parameters provides the received signal levels as [-77.2652, -86.8379, -92.4376, -96.4107, -99.4924] dBm at the above given distances.

Matlab Simulation:

```
%Matlab code to simulate Hata-Okumura Models
clc;clear all;
%----------Input Section----------
Hbts=  50 ;%Height measured from the base of the BTS tower to the radiation centerline
Tbts = 350 ;%Terrain elevation at the location of the BTS
Htav= 300;%Height of the average terrain (from 3 Km to 15 km distance from the BTS)
Hm=3 ;%Height of the mobile antenna in meters
f=870 ;%100:100:3000; %Range of frequencies in MHz
d=3:3:15; %Range of Tx-Rx separation distances in Kilometers
Pt = 19.5; %Power transmitted by the BTS antenna in Watts
Gt= 10; %BTS antenna gain in dBi

%---------------------------------------
Hb=Hbts+Tbts-Htav  ;%Effective Height of the BTS antenna in meters
%Cell array to store various model names
models = {'Big City (Urban model)';'Small & Medium City (Urban model)';'Sub-urban
environment';'Open Rural environment'};
display('Hata-Okumura Model');
display(['1 ' models{1,1}]);
display(['2 ' models{2,1}]);
display(['3 ' models{3,1}]);
display(['4 ' models{4,1}]);
reply = input('Select Your choice of environment : ','s');

if 0<str2num(reply)<4
    modelName = models{str2num(reply),1};
    display(['Chosen Model : ' modelName])
```

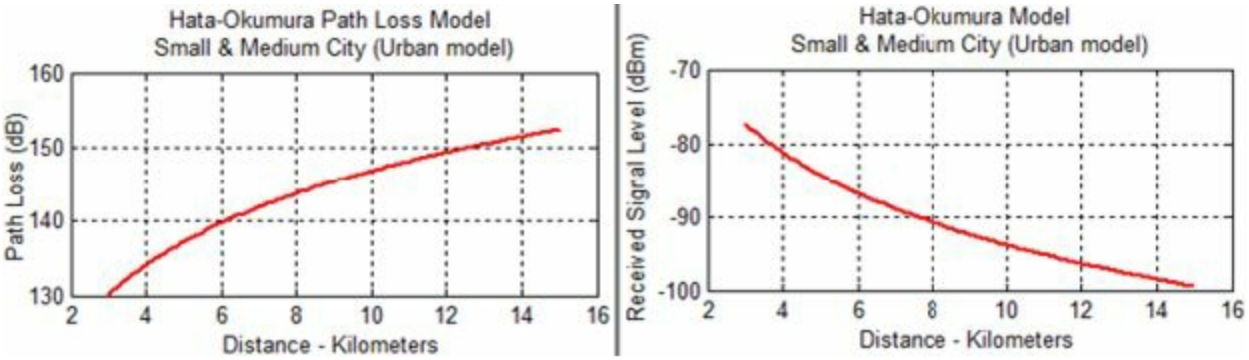
```
else
         error('Invalid Selection');
end
switch reply
    case '1',
        C=0;
        if f<=200
          aHm=8.29*(log10(1.54*Hm))^2-1.1;
        else
          aHm=3.2*(log10(11.75*Hm))^2-4.97;
        end
    case '2',
        C=0;
        aHm = (1.1*log10(f)-0.7)*Hm-(1.56*log10(f)-0.8);
    case '3',
        aHm = (1.1*log10(f)-0.7)*Hm-(1.56*log10(f)-0.8);
        C=-2*(log10(f/28))^2-5.4;
    case '4',
        aHm = (1.1*log10(f)-0.7)*Hm-(1.56*log10(f)-0.8);
        C=-4.78*(log10(f))^2+18.33*log10(f)-40.98;
    otherwise ,
        error('Invalid model selection');
end

A = 69.55 + 26.16*log10(f) - 13.82*log10(Hb)-aHm;
B = 44.9 - 6.55*log10(Hb);
PL=A+B*log10(d)+C;

subplot(2,1,1)
plot(d,PL,'r','LineWidth',2);
title(['Hata-Okumura Path Loss Model for : ' modelName]);
xlabel('Distance - Kilometers');
ylabel('Path Loss (dB)');

%Compute Received Signal Level
Pr = 10*log10(Pt*1000)+Gt-PL
subplot(2,1,2)
plot(d,Pr,'r','LineWidth',2);
title(['Hata-Okumura Model for : ' modelName]);
xlabel('Distance - Kilometers');
ylabel('Received Signal Level (dBm)');
```

Simulation Results:

>>Hata-Okumura Model:

1 Big City (Urban model)

2 Small & Medium City (Urban model)
3 Sub-urban environment
4 Open rural environment
Select your choice of environment: 2
Chosen Model: Small & Medium City (Urban model)
Pr = -77.2652 -86.8379 -92.4376 -96.4107 -99.4924



******

Back to Table of Contents

5.5 Introduction to Fading Models

In a wireless communication channel, the transmitted signal can travel from transmitter to receiver over multiple reflective paths or get obstructed by obstracles on the path. This gives rise to fluctuations in amplitude, phase and angle of arrival of the received signal. For example, the transmitted signal from the BTS (base transceiver station) may suffer multiple reflections from the buildings nearby, before reaching the mobile station. Fading causes fluctuations in the attenuation of the signal as it travels along the medium.

As seen in chapter 5.1, fading can be classified into two categories; slow fading/shadowing and fast fading/multipath induced fading. The classification can be expanded further by considering other factors or channel characteristics.

Understanding the physical effects in a wireless environment:

The physical effects observed in a wireless channel fall into following categories for fixed terminals (example: BTS) and mobile terminals (example: a moving receiver)

Effects on fixed terminals:
1. Path loss
2. Background noise

Effects on mobile terminals:
1. Path loss
2. Background noise
3. Doppler effect due to motion
4. Dispersion in Time or frequency selectivity

5. Channel impulse response varying with time which causes dispersion in frequency and amplitude fluctuations (fading) due to multipath interference.

Characterizing amplitude fluctuations:

As discussed in Chapter 5.1, three phenomenons namely 1) Propagation path loss 2) Shadowing/large scale fading 3) Multipath or small scale fading contribute to amplitude fluctuations in the received signal. It is often desirable to characterize the amplitude fluctuations statistically. Rayleigh, Rician and Nakagami distributions are some of the distributions that are used to model the amplitude fluctuations.

Characterizing Dispersion in Time:

In a multipath environment, the transmitted signal is reflected and scattered along the way by several obstacles. This leads to the phenomena called – "Dispersion in Time" or equivalently "frequency selectivity". If received signal arrives with different delays, it leads to time-dispersion of the transmitted signal. This is best characterized by Power Delay Profile or PDP. Two scalars namely Delay spread and coherence bandwidth are derived from the PDP.

Delay spread can be interpreted as the time difference between the arrival of first multipath component and the last multipath component. It is also quantified in Root-Mean-Squared (RMS) sense which gives the RMS delay spread.

Characterizing Dispersion in Frequency:

Usually, in the real world, the receiver and transmitter or the scatterers can be in relative motion to each other. Such scenario gives rise to Doppler Effect or "Dispersion in Frequency". This is caused by time varying (due to relative motion) nature of the channel. Dopple effect causes frequeny shift in the receiver and is called Doppler Spread. It characterizes the rate of fluctuations in fading and it depends on velocity. If a relative motion exists between the receiver and transmitter or the scatterers, the amplitude (fading level) of the received signal varies as a function of position. The phenomenon of dispersion in frequency is best described by Doppler spread and coherence time of the channel.

Types of fading models:

A model is necessary to predict the effects of this fading accurately in order to mitigate its effects. Some of the models used to model multipath fading are

1) Rayleigh Fading model (Clarke's model, Young's model)
2) Rician Fading model
3) Nakagami Fading model
4) Weibull Fading model
5) Log-Normal Shadowing model

Apart from multipath reflection there might also be dispersive time varying effects in the channel that is being modeled. One such effect is Doppler Shift that is caused when the receiver and/or transmitter is in motion with respect to each other. In such cases the dispersive effect is also modeled along with the chosen multipath model.

5.6 Rayleigh Fading and Rayleigh Distribution

The delays associated with different signal paths in a multipath fading channel change in an unpredictable manner and can only be characterized statistically. When there are a large number of paths, the central limit theorem can be applied to model the time-variant impulse response of the channel as a complex-valued Gaussian random process. When the impulse response is modeled as a zero-mean complex-valued Gaussian process, the channel is said to be a Rayleigh fading channel.

The model behind Rician fading is similar to that for Rayleigh fading, except that in Rician fading a strong dominant component is present. This dominant component can for instance be the line-of-sight wave.

Here the Rayleigh Fading model is assumed to have only two multipath components X(t) and Y(t). Rayleigh Fading can be obtained from zero-mean complex Gaussian processes (X(t) and Y(t)). Simply adding the two Gaussian Random variables and taking the square root (envelope) gives a single tap Rayleigh distributed process. The phase of such random variable follows uniform distribution.

Consider two Gaussian random variables with zero mean and same variance $X \sim N(0, \sigma^2)$ and $Y \sim N(0, \sigma^2)$.

Let's define a complex Gaussian random variable (to mimic IQ channel) as

$$Z = X + jY$$

Now, the envelope of the complex random variable is given by

$$R = \sqrt{X^2 + Y^2}$$

And the phase is given by

$$\varphi = tan^{-1}\left(\frac{Y}{X}\right)$$

The envelope follows Rayleigh distribution and the phase will be uniformly distributed

The probability density function (Rayleigh distribution) of the above mentioned amplitude response is given by

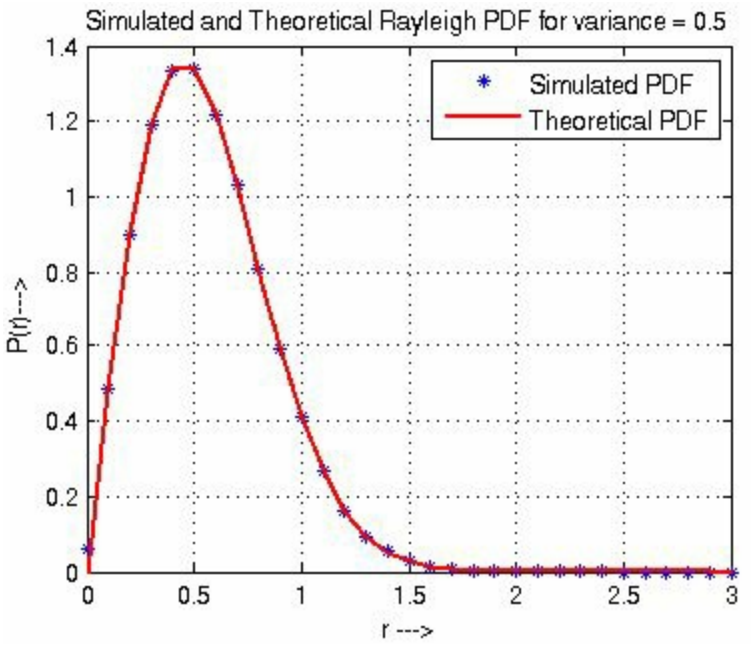
$$f(r) = \frac{r}{\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right)$$

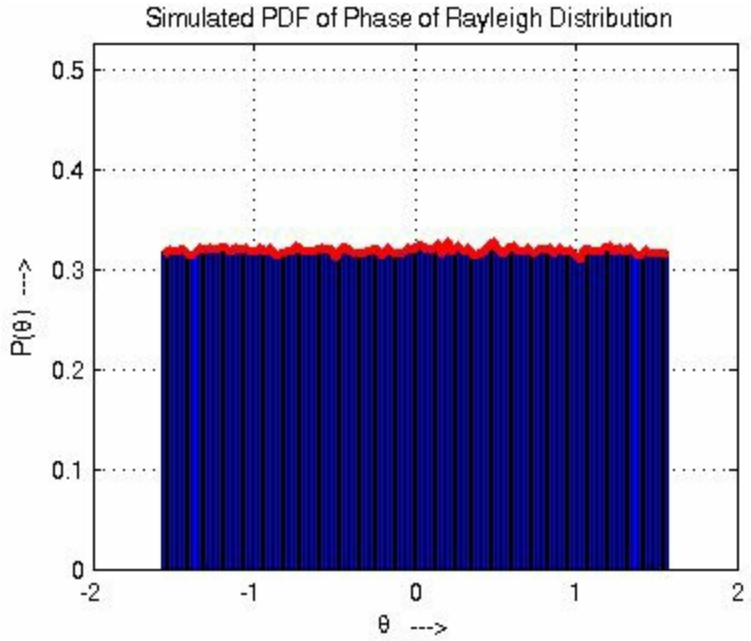
Matlab Code:

```
%----Rayleigh_PDF----------------------------------------
%----------Input Section----------------
N=1000000;  %Number of samples to generate
variance = 0.2; % Variance of underlying Gaussian random variables
%-------------------------------------
%Independent Gaussian random variables with zero mean and unit variance
x = randn(1, N);
y = randn(1, N);

%Rayleigh fading envelope with the desired variance
```

```
r = sqrt(variance*(x.^2 + y.^2));

%Define bin steps and range for histogram plotting
step = 0.1; range = 0:step:3;

%Get histogram values and approximate it to get the pdf curve
h = hist(r, range);
approxPDF = h/(step*sum(h)); %Simulated PDF from the x and y samples
%Theoritical PDF from the Rayleigh Fading equation
theoretical = (range/variance).*exp(-range.^2/(2*variance));
plot(range, approxPDF,'b*', range, theoretical,'r');
title('Simulated and Theoretical Rayleigh PDF for variance = 0.5')
legend('Simulated PDF','Theoretical PDF')
xlabel('r --->');
ylabel('P(r)---> ');
grid;

%PDF of phase of the Rayleigh envelope
theta = atan(y./x);
figure(2)
hist(theta); %Plot  histogram of the phase part

%Approximate the histogram of the phase part to a nice PDF curve
[counts,range] = hist(theta,100);
step=range(2)-range(1);
%Normalizing the PDF to match theoretical curve
approxPDF = counts/(step*sum(counts)); %Simulated PDF from the x and y samples
bar(range, approxPDF,'b');
hold on
plotHandle=plot(range, approxPDF,'r');
set(plotHandle,'LineWidth',3.5);
axis([-2 2 0 max(approxPDF)+0.2])
hold off
title('Simulated PDF of Phase of Rayleigh Distribution ');
xlabel('\theta   --->');
ylabel('P(\theta)  --->');
grid;
```

Here two independent identically distributed (i.i.d) Gaussian random arrays are generated using "randn" function in Matlab and the envelope of their sum is computed to give Rayleigh Fading process. Histogram is used to plot the PDF of the generated process and its phase plot is also drawn. From the plots it is evident that the envelope of the random variable (created by adding two i.i.d Gaussian random variables) follows Rayleigh distribution and the phase follows uniform distribution.





******

Back to Table of Contents

5.7 Rayleigh Fading Simulation – Young's model

In the previous section of text, a simple fading scenario using two multipaths was discussed. Rayleigh Fading channel with Doppler shift is considered in this article.

Consider a channel affected by both Rayleigh Fading phenomena (non-line-of sight multipaths – each behaving as i.i.d Gaussian random variables) and Doppler Shift. Rayleigh Fading is caused due to multipath reflections of the received signal before it reaches the receiver and the Doppler Shift is caused due to the difference in the relative velocity/motion between the transmitter and the receiver. This scenario is encountered in day to day mobile communications.

A number of simulation algorithms are proposed for generation of correlated Rayleigh random variables. David J.Young and Norman C Beaulieu proposed a method in their paper titled "The Generation of Correlated Rayleigh Random Variates by Inverse Discrete Fourier Transform" [Young2000] based on the inverse discrete Fourier transform (IDFT). It is a modification of the Smith's algorithm [Smith1975] which is normally used for Rayleigh fading simulation. This method requires exactly one-half the number of IDFT operations and roughly two-thirds the computer memory of the original method – as the authors of the paper claim.

Rayleigh Fading can be simulated by adding two Gaussian Random variables as mentioned in my previous post. The effect of Doppler shift is incorporated by modeling the Doppler Effect as a frequency domain filter.

The model proposed by Young et.al [Young2000] is shown below.

The "Fading effect + Doppler Shift" is simulated by multiplying the Gaussian Random variables and the Doppler Shift's Frequency domain representation. Then IDFT is performed to bring them into time domain representation. The Doppler Filter used to represent the Doppler Shift effect is derived in Young's paper.

The equation for the Doppler Filter is:

$$F_M[k] = \begin{cases} 0, & k = 0 \\ \sqrt{\dfrac{1}{2\sqrt{1-\left(\frac{k}{Nf_m}\right)^2}}}, & k = 1,2,\dots,k_m-1 \\ \sqrt{\dfrac{k_m}{2}\left[\dfrac{\pi}{2}-arctan\left(\dfrac{k_m-1}{\sqrt{2k_m-1}}\right)\right]}, & k = k_m \\ 0, & k = k_m+1,\dots,N-k_m-1 \\ \sqrt{\dfrac{k_m}{2}\left[\dfrac{\pi}{2}-arctan\left(\dfrac{k_m-1}{\sqrt{2k_m-1}}\right)\right]}, & k = N-k_m \\ \sqrt{\dfrac{1}{2\sqrt{1-\left(\frac{N-k}{Nf_m}\right)^2}}}, & k = N-k_m+1,\dots,N-2,N-1 \end{cases}$$

Matlab Code

File_1: dopplerFilter.m

```
function [freqResponse]=dopplerFilter(Fm,Fs,M)

F = zeros(1,M);
dopplerRatio = Fm/Fs;
km=dopplerRatio*M;

for i=1:M
    if (i==1),
```

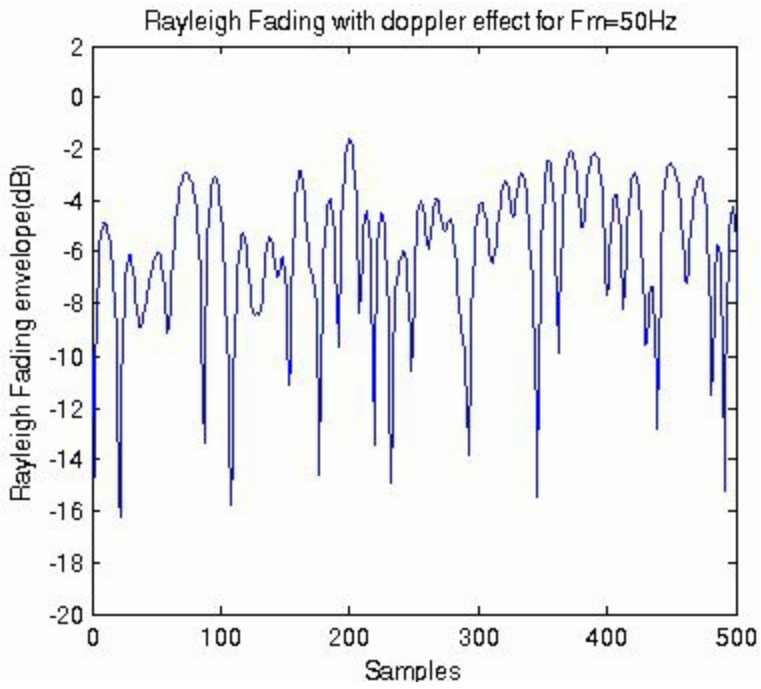
```
        F(i)=0;
    elseif (i>=2 && i<=km),
        F(i)=sqrt(1/(2*sqrt(1-(i/(M*dopplerRatio)^2))));
    elseif (i==km+1),
        F(i)=sqrt(km/2*(pi/2-atan((km-1)/sqrt(2*km-1))));
    elseif (i>=km+2 && i<=M-km+2),
        F(i) = 0;
    elseif (i==M*km),
        F(i)=sqrt(km/2*(pi/2-atan((km-1)/sqrt(2*km-1))));
    else
        F(i)=sqrt(1/(2*sqrt(1-((M-i)/(M*dopplerRatio)^2))));
    end
end
freqResponse = F;
```

File_2: testYoungsModel.m

```
%-------Input Section-----------------------------------
N=4096; %Number of sample points (N)
%N is usually a power of 2
Fm=50; %Maximum Doppler Frequency Shift
Fs=1000; %Sampling Frequency
%Baseband Gaussian Noise Generators
mean = 0; %Mean of  Gaussian random variables
variance = 0.5; %Variance of Gaussian random variables

%-----------------------------------------------------
sdev = sqrt(variance); %Standard Deviation of Gaussian RV
%In-phase Noise components
G1 = mean + sdev.*randn(1,N) ; %N i.i.d Gaussian random samples
%Quadrature-phase Noise components
G2 = mean + sdev.*randn(1,N) ; %N i.i.d Gaussian random samples
C = G1-1i*G2;

%Define Spectral characteristics of the Doppler effect in frequency domain Fk = Doppler Filter
output
Fk = dopplerFilter(Fm,Fs,N);

%Multiply C by filter sequency Fk
U = C.*Fk;
NFFT = 2^nextpow2(length(U));
u=abs(ifft(U,NFFT)); %Take IDFT
normalizedFading = u/max(u); %Baseband Rayleigh envelope

plot(10*log10(normalizedFading)) %plot command
title(['Rayleigh Fading with doppler effect for Fm=',num2str(Fm),'Hz']);
```

```
xlabel('Samples');
ylabel('Rayleigh Fading envelope(dB)');
axis([0 500 -20 2]); %showing only few samples for clarity
```

Simulation Result:



******

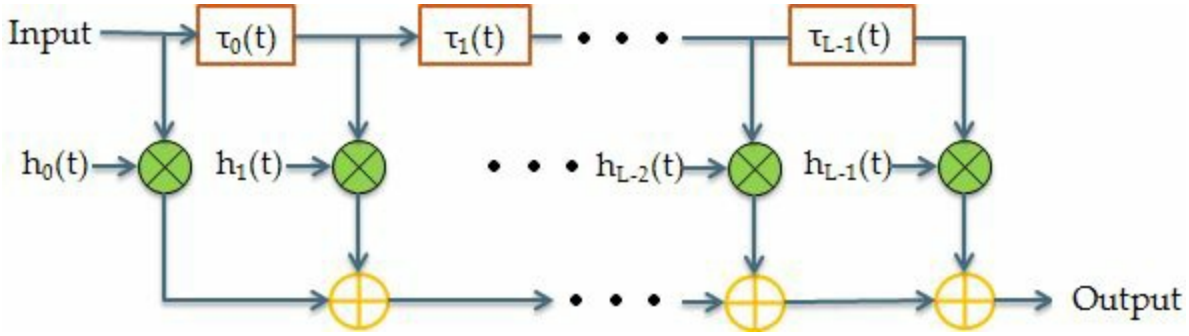
Back to Table of Contents

5.8 Simulation of Rayleigh Fading Model - (Clarke's Model - Sum of Sinusoids)

A multipath fading channel can be modeled as an FIR (Finite Impulse Response)-filter with the following impulse response.

$$h(\tau; t) = h_0(t)\delta(\tau - \tau_0(t)) + h_1(t)\delta(\tau - \tau_1(t)) + \cdots + h_{L-1}(t)\delta(\tau - \tau_{L-1}(t))$$

where h(τ,t) is the time varying impulse response of the multipath fading channel having L multi-paths and hi(t) and τi(t) denote the time varying complex gain and excess delay of the i-th path. The above mentioned impulse response can be implemented as an FIR filter as shown below:



The channel under consideration can be modeled as a multipath fading channel in which the impulse response may follow distributions like Rayleigh distribution (in which there is no Line of Sight

(LOS) ray between transmitter and receiver) or as Rician distribution (dominant LOS path exist between transmitter and receiver), Nagami distribution, Weibull distribution etc.

Different methods of simulation techniques were proposed to simulate/model multipath channels. Some of the models include Clarke's reference model, Jake's model, Young's model, filtered Gaussian noise model etc.

A Rayleigh fading channel (flat fading channel) is considered in this text. For simplicity we fix the excess delays $\tau_i(t)$ in the above equation and we generate $h_i(t)$ that follows Rayleigh distribution. In this simulation Clarke's Rayleigh fading model is used. This model is also called mathematical reference model and is commonly considered as a computationally inefficient model compared to Jake's Rayleigh Fading simulator.

Theory of Rayleigh Fading:

Let us denote the complex impulse response h(t) of the flat fading channel as follows :

$$h(t) = h_I(t) + jh_Q(t)$$

where $h_I(t)$ and $h_Q(t)$ are zero mean Gaussian distributed. Therefore the fading envelope is Rayleigh distributed and is given by

$$|h(t)| = \sqrt{|h_I(t)|^2 + |h_Q(t)|^2}$$

The probability density function (Rayleigh distribution) of the above mentioned amplitude response is given by

$$f(z) = \frac{2z}{\sigma^2}\exp\left(-\frac{z^2}{\sigma^2}\right), \text{ where } \sigma^2 = \mathrm{E}[|h(t)|^2]$$

We will use the Clarke's Rayleigh Fading model (given below) and check the statistical properties of the random process generated by the model against the statistical properties of Rayleigh distribution.

Clarke's Rayleigh Fading model:

The random process of flat Rayleigh fading with M multi-paths can be simulated with the sum-of-sinusoid method described as

$$h_I(nT_s) = \frac{1}{\sqrt{M}}\sum_{m=1}^{M} cos\left\{2\pi f_D\, cos\left[\frac{(2m-1)\pi+\theta}{4M}\right].nT_s + \alpha_m\right\}$$
$$h_Q(nT_s) = \frac{1}{\sqrt{M}}\sum_{m=1}^{M} sin\left\{2\pi f_D\, cos\left[\frac{(2m-1)\pi+\theta}{4M}\right].nT_s + \beta_m\right\}$$
$$h(nT_s) = h_I(nT_s) + jh_Q(nT_s)$$

$$where\ \theta,\ \alpha_m\ and\ \beta_m\ are\ uniformly\ distributed\ over\ [0, 2\pi)$$
$$for\ all\ n\ and\ are\ mutually\ distributed,$$
$$f_D = maximum\ Doppler\ spread,$$
$$T_s = sampling\ period,\ n = sample\ index$$

Simulation:

1) The Rayleigh fading model is implemented as a function in Matlab with following parameters:

M=number of multi-paths in the fading channel, N = number of samples to generate, f_d=maximum Doppler spread in Hz, T_S = sampling period.

File_1: rayleighFading.m

```
function [h]=rayleighFading(M,N,fd,Ts)
% function to generate Rayleigh Fading samples based on Clarke's model
% M = number of multi-paths in the channel
% N = number of samples to generate
% fd = maximum Doppler frequency
% Ts = sampling period
a=0;
b=2*pi;
alpha=a+(b-a)*rand(1,M); %uniformly distributed from 0 to 2 pi
beta=a+(b-a)*rand(1,M); %uniformly distributed from 0 to 2 pi
theta=a+(b-a)*rand(1,M); %uniformly distributed from 0 to 2 pi
m=1:M;
for n=1:N;
    x=cos(((2.*m-1)*pi+theta)/(4*M));
    h_re(n)=1/sqrt(M)*sum(cos(2*pi*fd*x*n'*Ts+alpha));
    h_im(n)=1/sqrt(M)*sum(sin(2*pi*fd*x*n'*Ts+beta));
end
h=h_re+j*h_im;
end
```

2) A test program is coded which uses the above mentioned function is used to generate Rayleigh Fading samples with the following values for the function arguments. M=15; N=10^5; fd=100 Hz;Ts=0.0001 second;

File_2: testRayleighFading.m

```
clc;clear;
M=15; %number of multipaths
N=10^5; %number of samples to generate
fd=100; % Maximum doppler spread in hertz
Ts=0.0001; % Sampling period in seconds

h=rayleighFading(M,N,fd,Ts);
h_re=real(h);
h_im=imag(h);

figure;
subplot(2,1,1);
plot([0:N-1]*Ts,h_re);
title('Real part of impulse response of the Flat Fading channel');
xlabel('time(s)');ylabel('Amplitude |hI(t)|');
```

```
subplot(2,1,2);
plot([0:N-1]*Ts,h_im);
title('Imaginary part of impulse response of the Flat Fading channel');
xlabel('time(s)');ylabel('Amplitude |hQ(t)|');
figure;

subplot(2,1,1);
plot([0:N-1]*Ts,10*log10(abs(h)));
title('Amplitude Response of the Flat Fading channel');
xlabel('time(s)');ylabel('Magnitude |h(t)|');
subplot(2,1,2);
plot([0:N-1]*Ts,angle(h));
title('Phase response of the Flat Fading channel');
xlabel('time(s)');ylabel('Phase angle(h(t))');

%Statistical properties
mean_re=mean(h_re)
mean_im=mean(h_im)
var_re=var(h_re)
var_im=var(h_im)

%comparing the PDF of real part of generated samples against the PDF of Gaussian distribution
[val,bin]=hist(h_re,1000); % pdf of real part of generated Raleigh Fading Samples
figure;
plot(bin,val/trapz(bin,val)); %Normalizing the PDF to match theoretical result
%Trapz function gives the total area under the PDF curve. It is used as the normalization factor
hold on;
x=-2:0.1:2;
y=normpdf(x,0,sqrt(0.5)); % theoretical gaussian pdf
plot(x,y,'r');
title('Probability density function');
legend('Simulated pdf','Theoretical Gaussian pdf');

figure;
%comparing the PDF of overal response of the channel against the PDF of Rayleigh distribution
[val,bin]=hist(abs(h),1000); % pdf of generated Raleigh Fading samples
plot(bin,val/trapz(bin,val)); %Normalizing the PDF to match theoretical result
%Trapz function gives the total area under the PDF curve. It is used as the normalization factor
hold on;
z=0:0.1:3; sigma=1;
y=2*z/(sigma^2).*exp(-z.^2/(sigma^2)); % theoretical Raleigh pdf
plot(z,y,'r');
title('Probability density function');
legend('Simulated pdf','Theoretical Rayleigh pdf');
```

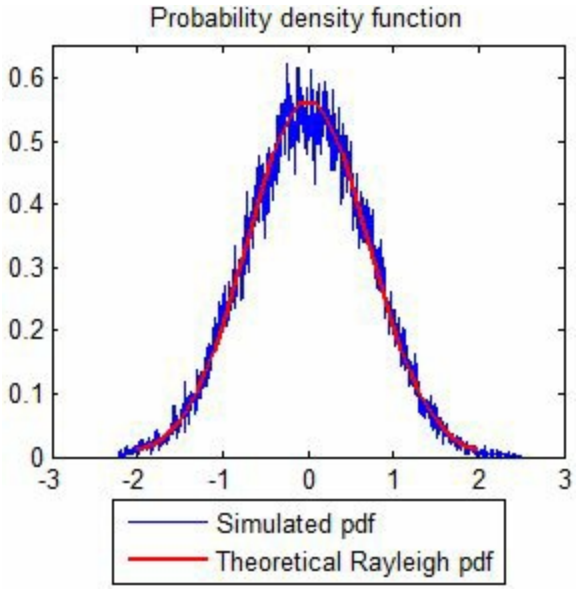
Investigation of Statistical Properties of samples generated using Clarke's model:

3) From the above test program, the mean and variance of the real and imaginary parts of generated samples are

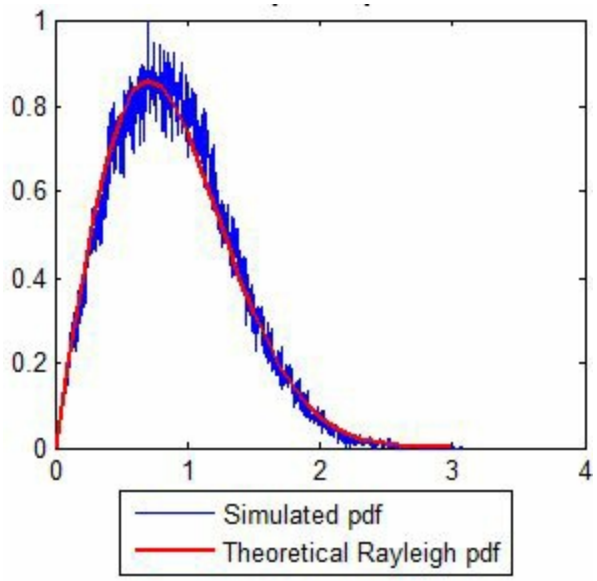
Mean of real part ~=0
Mean of imaginary part ~=0
Variance of real part = 0.4989 ~=0.5
Variance of imaginary part = 0.4989 ~=0.5

The results imply that the mean of the real and imaginary parts are same and are equal to zero. The variances of the real and imaginary parts are approximately equal to 0.5.

4) Next the PDF of the real part of the simulated samples are plotted and compared against the pdf of Gaussian distribution (with mean=0 and variance =0.5)

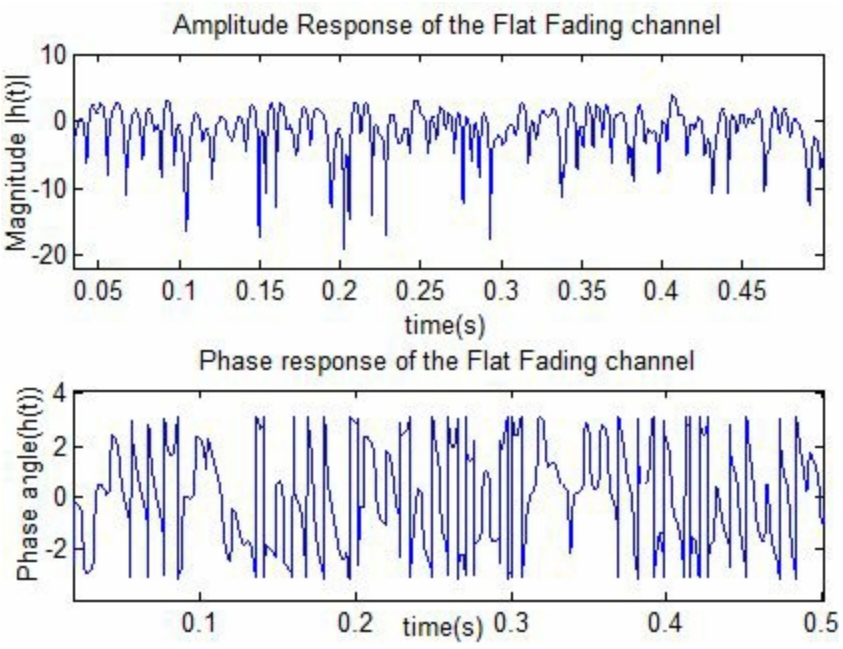


5) The PDF of the generated Rayleigh fading samples are plotted and compared against PDF of Rayleigh distribution (with variance=1)



6) From 4) and 5) we confirm that the samples generated by Clarke's model follows Rayleigh distribution (with variance = 1) and the real and imaginary part of the samples follow Gaussian distribution (with mean=0 and variance =0.5).

7) The Magnitude and Phase response of the generated Rayleigh Fading samples are plotted here.



******

Back to Table of Contents

5.9 Rician Fading and Rician Distribution

The model behind Rician fading is similar to that of Rayleigh fading, except that in Rician fading a strong dominant component is present. This dominant component can for instance be the line-of-sight wave. This is modeled by using two Gaussian random variables – one with zero mean and another with non-zero mean.

Consider two Gaussian random variables $X \sim N(m_1, \sigma^2)$ and $Y \sim N(m_2, \sigma^2)$. Here m_1 and m_2 are the means of the distributions and σ^2 is the variance.

Let us define a complex Gaussian random variable (to mimic IQ channel) as

$$Z = X + jY$$

Now, the envelope of the complex random variable is given by

$$R = \sqrt{X^2 + Y^2}$$

And the phase is given by

$$\varphi = tan^{-1}\left(\frac{Y}{X}\right)$$

Since, the two variables X and Y have different "means", a non-centrality parameter (indicating the non-central mean) is defined.

$$s = \sqrt{m_1^2 + m_2^2}$$

The non-centrality parameter (the imbalance in the means) is caused by the presence of dominant path

in a Rician Fading environment. Due to this, the Rician K factor - representing the ratio of power of Line-Of-Sight (LOS) (or dominant multipath component) and the power of Non-Line-Of-Sight (NLOS) (or the remaining multipath components) is defined in such scenario.

$$\kappa = \frac{Power\ of\ LOS\ component}{Power\ of\ NLOS\ components}$$

Statistically, this can be represented as the power in the faded envelope that has been produced by the means of X and Y.

$$\kappa = \frac{m_1^2 + m_2^2}{2\sigma^2} = \frac{s^2}{2\sigma^2}$$

The envelope (R) follows Rician distribution, whose PDF is given by

$$f_R(r) = \frac{r}{\sigma^2} exp\left(-\frac{r^2 + s^2}{2\sigma^2}\right) I_0\left(\frac{rs}{\sigma^2}\right)$$

Where, $I_0(x)$ is the modified zero-th order Bessel function of the first kind.

Simulating Rician PDF:

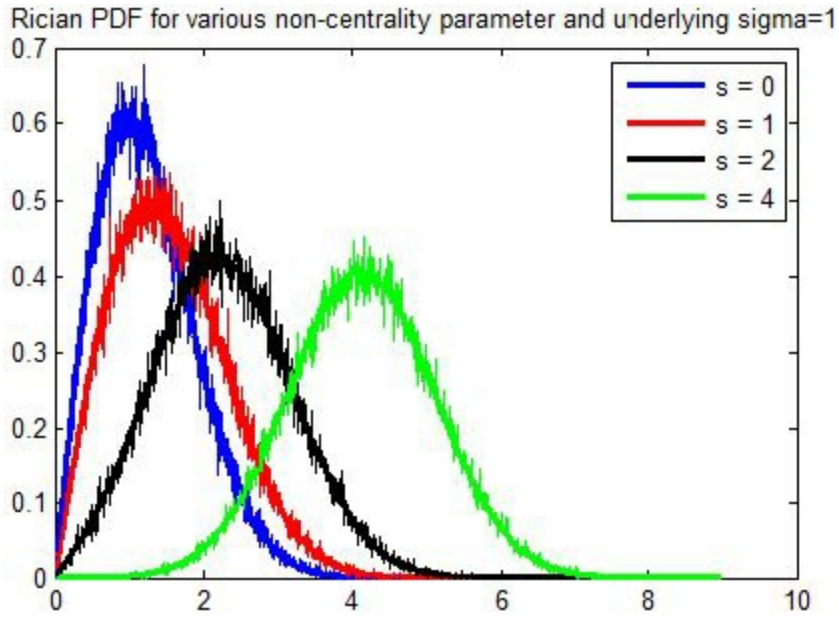
In the following simulation, Rician fading envelope is simulated for various non-centrality parameter(s) and $\sigma^2=1$ for the underlying Gaussian random variables X and Y. The random variables X & Y used to generate Rician distribution differ only in their means. Their variances remain the same. For convenience, one of the Gaussian random variable (say X) is generated with mean=s (the non centrality parameter) and variance= σ^2. The other random variable (say Y) is generated with mean=0 and variance= σ^2.

Note that (m_1, m_2 or s) and σ^2 are not the means and variance of the Rician distribution.

The following Matlab code uses this concept to simulate a Rician PDF for various non-centrality parameter(s) and sigma=1 for the underlying Gaussian random variable. Theoretical PDF curves are superimposed on the simulated curves for verification.

```
N=100000; %Number of Samples
sigma=1; %Variance of underlying Gaussian Variables
s=[0 1 2 4]; %Non-Centrality Parameter

plotStyle={'b-','r-','k-','g-'};

%Simulating the PDF from two Gaussian Random Variables
for i = 1: length(s)

    X = s(i) + sigma.*randn(1,N); %Gaussian RV with mean=s and given sigma
    Y = 0 + sigma.*randn(1,N); %Gaussian RV with mean=0 and same sigma as Y

    Z=X+1i*Y;
    [val,bin]=hist(abs(Z),1000); % pdf of generated Raleigh Fading samples
    plot(bin,val/trapz(bin,val),plotStyle{i}); %Normalizing the PDF to match theoretical result
```

```
    %Trapz function gives the total area under the PDF curve. It is used as the normalization factor

    hold on;
end

%Theoretical PDF computation
for i=1:length(s)
    x=s(i);
    m1=sqrt(x);
    m2=sqrt(x*(x-1));

    r=0:0.01:9;
    ss=sqrt(m1^2+m2^2);
    x=r.*ss/(sigma^2);
    f=r./(sigma^2).*exp(-((r.^2+ss^2)./(2*sigma^2))).*besseli(0,x);

    plot(r,f,plotStyle{i},'LineWidth',2.5);
    legendInfo{i} = ['s = ' num2str(s(i))];

    hold on;
end
legend(legendInfo);
```



******

Back to Table of Contents

Chapter 6: Digital Modulations

6.1 BPSK Modulation and Demodulation

BPSK Modulation:

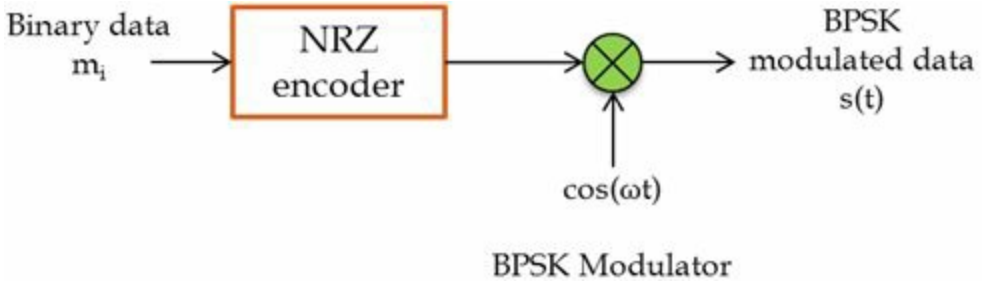
In digital modulation techniques, a set of basis functions are chosen for a particular modulation scheme. Generally the basis functions are orthogonal to each other. Basis functions can be derived using 'Gram Schmidt orthogonalization' procedure [Trefethen1997]. Once the basis functions are chosen, any vector in the signal space can be represented as a linear combination of the basis functions.

In Binary Phase Shift Keying (BPSK) only one sinusoid is taken as basis function modulation. Modulation is achieved by varying the phase of the basis function depending on the message bits. The following equation outlines BPSK modulation technique.

$$S_0(t) = A\cos(\omega t) \quad \Rightarrow represents\ '0'$$
$$S_1(t) = A\cos(\omega t + \pi) \quad \Rightarrow represents\ '1'$$

Here, A is the amplitude of the sinusoidal signal, ω is the carrier angular frequency measured in radians/second, t is the instantaneous time. $S_0(t)$ represents the carrier signal when '0' is transmitted and $S_1(t)$ represents the carrier signal when '1' was transmitted. The constellation diagram of BPSK will show the constellation points lying entirely on the x axis. It has no projection on the y axis. This means that the BPSK modulated signal will have an in-phase component (I) but no quadrature component (Q). This is because it has only one basis function.

A BPSK modulator can be implemented by coding the message bits using NRZ coding (1 represented by positive voltage and 0 represented by negative voltage) and multiplying the output by a reference oscillator running at carrier frequency ω.

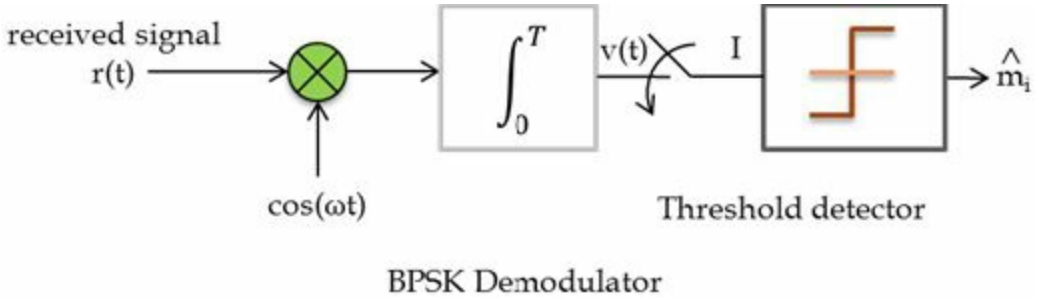


BPSK Modulator

BPSK Demodulation:

For BPSK demodulator, a coherent demodulator is taken as an example. In coherent detection technique the knowledge of the carrier frequency and phase must be known to the receiver. This can be achieved by using a Costas loop or a Phase Lock Loop (PLL) at the receiver. A PLL essentially locks to the incoming carrier frequency and tracks the variations in frequency and phase. For the following simulation, neither a PLL nor a Costas loop is used but instead the output of the PLL or Costas loop is used. For demonstration purposes we simply assume that the carrier phase recovery is done and directly use the generated reference frequency at the receiver -cos(ωt).

In the demodulator the received signal is multiplied by a reference frequency generator (assuming the PLL/Costas loop is present). The multiplied output is integrated over one bit period using an integrator. A threshold detector makes a decision on each integrated bit based on a threshold. Since an NRZ signaling format is used with equal amplitudes in positive and negative direction, the threshold for this case would be '0'.



Matlab Code:

File 1: NRZ_Encoder.m

```
% NRZ_Encoder Line codes encoder
%[time,output,Fs]=NRZ_Encoder(input,Rb,amplitude,style)
%NRZ_Encoder NRZ encoder
%Input a stream of bits and specify bit-Rate, amplitude of the output signal and the style of encoding
Currently 3 encoding styles are supported namely 'Manchester','Unipolar'and 'Polar'.Outputs the NRZ
stream
function [time,output,Fs]=NRZ_Encoder(input,Rb,amplitude,style)

Fs=16*Rb; %Sampling frequency ,
oversampling factor= 32
Ts=1/Fs; % Sampling Period
Tb=1/Rb; % Bit period
output=[];

switch lower(style)
    case {'manchester'}
        for count=1:length(input),
          for tempTime=0:Ts:Tb/2-Ts,
           output=[output (-1)^(input(count))*amplitude];
          end
          for tempTime=Tb/2:Ts:Tb-Ts,
           output=[output (-1)^(input(count)+1)*amplitude];
          end
        end
    case {'unipolar'}
        for count=1:length(input),
          for tempTime=0:Ts:Tb-Ts,
           output=[output input(count)*amplitude];
          end
        end
    case {'polar'}
        for count=1:length(input),
          for tempTime=0:Ts:Tb-Ts,
           output=[output amplitude*(-1)^(1+input(count))];
          end
```

```
        end
    otherwise,
        disp('NRZ_Encoder(input,Rb,amplitude,style)-Unknown method given as "style" argument');
        disp('Accepted Styles are "Manchester", "Unipolar" and "Polar"');
    end
time=0:Ts:Tb*length(input)-Ts;
```

File 2: BPSK_Mod_Demod.m

%Demonstration of BPSK Modulation and Demodulation

%This scheme uses a simple BPSK modulation scheme in which a random binary stream is represented in NRZ format and the resulting signal is multiplied by reference carrier frequency. AWGN channel noise generated according to the desired noise variance is added with the BPSK modulated signal. The noise added signal is demodulated by multiplying it with the reference frequency and integrated over one bit period.

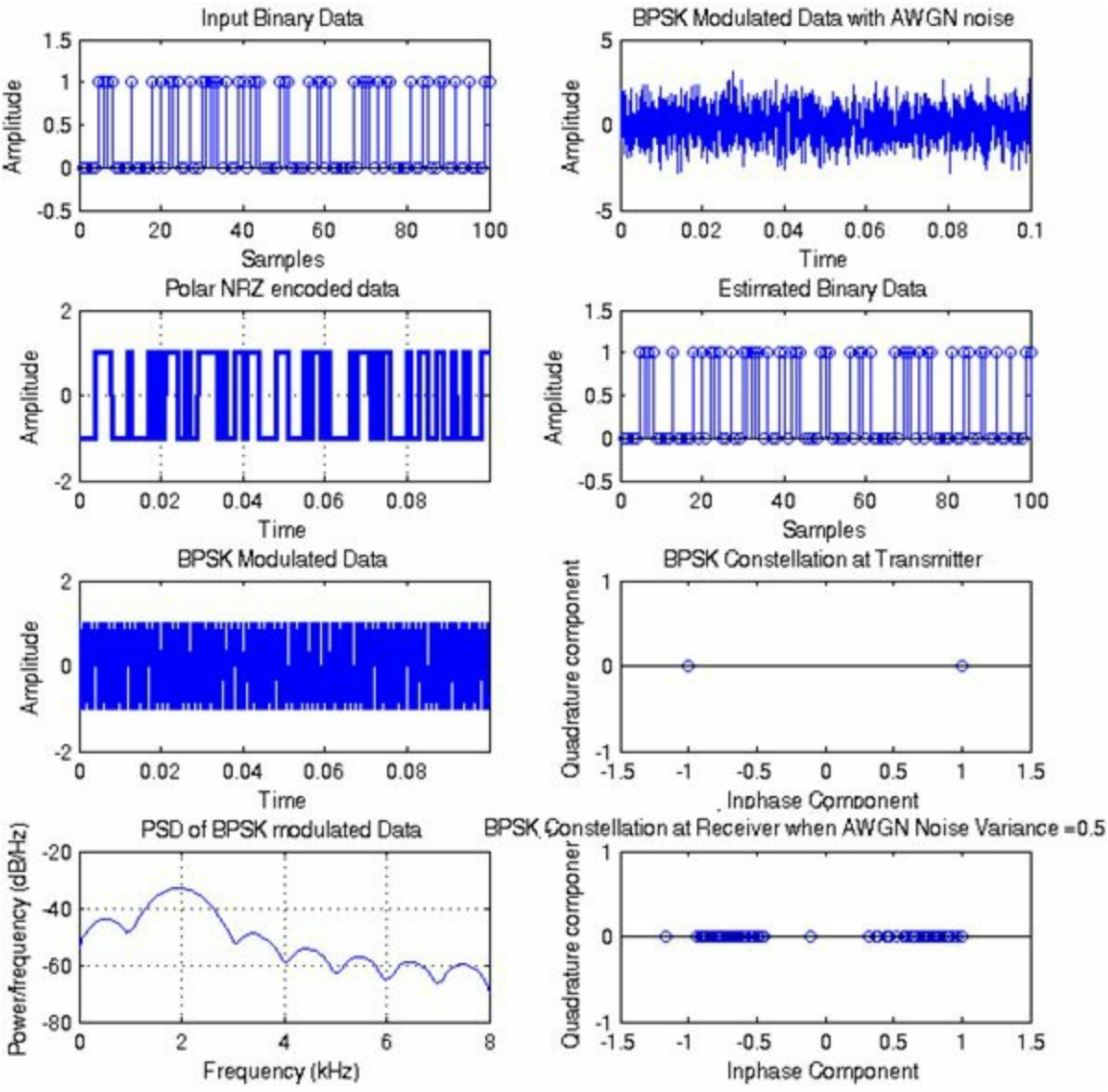
```
%clear; %clear all stored variables
N=100; %number of data bits
noiseVariance = 0.5; %Noise variance of AWGN channel
data=randn(1,N)>=0; %Generate uniformly distributed random data
Rb=1e3; %bit rate
amplitude=1; % Amplitude of NRZ data
[time,nrzData,Fs]=NRZ_Encoder(data,Rb,amplitude,'Polar');
Tb=1/Rb;

subplot(4,2,1);
stem(data);
xlabel('Samples');
ylabel('Amplitude');
title('Input Binary Data');
axis([0,N,-0.5,1.5]);

subplot(4,2,3);
plotHandle=plot(time,nrzData);
xlabel('Time');
ylabel('Amplitude');
title('Polar NRZ encoded data');
set(plotHandle,'LineWidth',2.5);
maxTime=max(time);
maxAmp=max(nrzData);
minAmp=min(nrzData);
axis([0,maxTime,minAmp-1,maxAmp+1]);
grid on;

Fc=2*Rb;
osc = sin(2*pi*Fc*time);
```

```
%BPSK modulation
bpskModulated = nrzData.*osc;

subplot(4,2,5);
plot(time,bpskModulated);
xlabel('Time');
ylabel('Amplitude');
title('BPSK Modulated Data');
maxTime=max(time);
maxAmp=max(nrzData);
minAmp=min(nrzData);
axis([0,maxTime,minAmp-1,maxAmp+1]);

%plotting the PSD of BPSK modulated data
subplot(4,2,7);

h=spectrum.welch; %Welch spectrum estimator
Hpsd = psd(h,bpskModulated,'Fs',Fs);
plot(Hpsd);
title('PSD of BPSK modulated Data');

%----------------------------------------
%Adding Channel Noise
%----------------------------------------
noise = sqrt(noiseVariance)*randn(1,length(bpskModulated));
received = bpskModulated + noise;
subplot(4,2,2);
plot(time,received);
xlabel('Time');
ylabel('Amplitude');
title('BPSK Modulated Data with AWGN noise');

%----------------------------------------
%BPSK Receiver
%----------------------------------------
%Multiplying the received signal with reference Oscillator
v = received.*osc;
%Integrator
integrationBase = 0:1/Fs:Tb-1/Fs;
for i = 0:(length(v)/(Tb*Fs))-1,
    y(i+1)=trapz(integrationBase,v(int32(i*Tb*Fs+1):int32((i+1)*Tb*Fs)));
end

%Threshold Comparator
```

```
estimatedBits=(y>=0);
subplot(4,2,4);
stem(estimatedBits);
xlabel('Samples');
ylabel('Amplitude');
title('Estimated Binary Data');
axis([0,N,-0.5,1.5]);

%----------------------------------------
%Bit Error rate Calculation
BER = sum(xor(data,estimatedBits))/length(data);

%Constellation Mapper at Transmitter side
subplot(4,2,6);
Q = zeros(1,length(nrzData)); %No Quadrature Component for BPSK
stem(nrzData,Q);
xlabel('Inphase Component');
ylabel('Quadrature Phase component');
title('BPSK Constellation at Transmitter');
axis([-1.5,1.5,-1,1]);

%constellation Mapper at receiver side
subplot(4,2,8);
Q = zeros(1,length(y)); %No Quadrature Component for BPSK
stem(y/max(y),Q);
xlabel('Inphase Component');
ylabel('Quadrature Phase component');
title(['BPSK Constellation at Receiver when AWGN Noise Variance =',num2str(noiseVariance)]);
axis([-1.5,1.5,-1,1]);
```

Simulation Results:

The output of the Matlab code provides more insight into the modulation techniques. Apart from plotting the modulated and demodulated signal it also shows the constellation at transmitter/receiver and the Power Spectral Density of the BPSK modulated signal.



******

Back to Table of Contents

6.2 BER vs. E_b/N_0 for BPSK modulation over AWGN

In the previous section we saw about how pass band BPSK modulation and demodulation can be done. This concept is extended further to simulate the performance of BPSK modulation technique over an AWGN. Note that this is a baseband simulation of BPSK modulation and demodulation. Baseband simulation is faster and yields performance results same as that of pass band simulation.

Transmitter:

For the BPSK modulation, a series of binary input message bits are generated of which 1′s are represented by 1V and 0′s are translated as -1V (equivalent to NRZ coding as discussed in the previous chapter).

AWGN channel:

For BPSK modulation the channel can be modeled as

$$y = ax + n$$

Where, y is the received signal at the input of the BPSK receiver, x is the modulated signal transmitted through the channel, 'a' is the channel amplitude scaling factor for the transmitted signal usually assumed as unity. 'n' is the Additive Gaussian White Noise random variable with zero mean and variance σ^2.

For an AWGN channel, the noise variance in terms of noise power spectral density (N_0) is given by

$$\sigma^2 = \frac{N_0}{2}$$

For M-ARY modulation schemes like M-PSK including BPSK, the symbol energy is given by

$$E_S = R_m R_c E_b$$

Where, E_S =Symbol energy per modulated bit (x), $R_m = \log_2(M)$, (for BPSK M=2, QPSK M=4, 16 QAM M=16 etc..,). R_C is the code rate of the system if a coding scheme is used. In this example, since no coding scheme is used $R_c = 1$. E_b is the Energy per information bit.

Assuming E_S=1 for BPSK (Symbol energy normalized to 1) E_b/N_0 can be represented as (using above equations),

$$\frac{E_b}{N_0} = \frac{E_S}{R_m R_c N_0} = \frac{E_S}{R_m R_c 2\sigma^2} = \frac{1}{2R_m R_c \sigma^2}$$

From the above equation the noise variance for the given E_b/N_0 can be calculated as

$$\sigma^2 = \left[2R_m R_c \frac{E_b}{N_0}\right]^{-1}$$

For the channel model "randn" function in Matlab is used to generate the noise term. This function generates noise with unit variance and zero mean. In order to generate a noise with sigma σ for the given E_b/N_0 ratio, use the above equation, find σ, multiply the "randn" generated noise with this sigma , add this final noise term with the transmitted signal to get the received signal.

Receiver:

BPSK receiver can be a simple threshold detector which categorizes the received signal as '0' or '1' depending on the threshold that is being set. Calculation of Theoretical BER for BPSK over AWGN is in Chapter 6.12

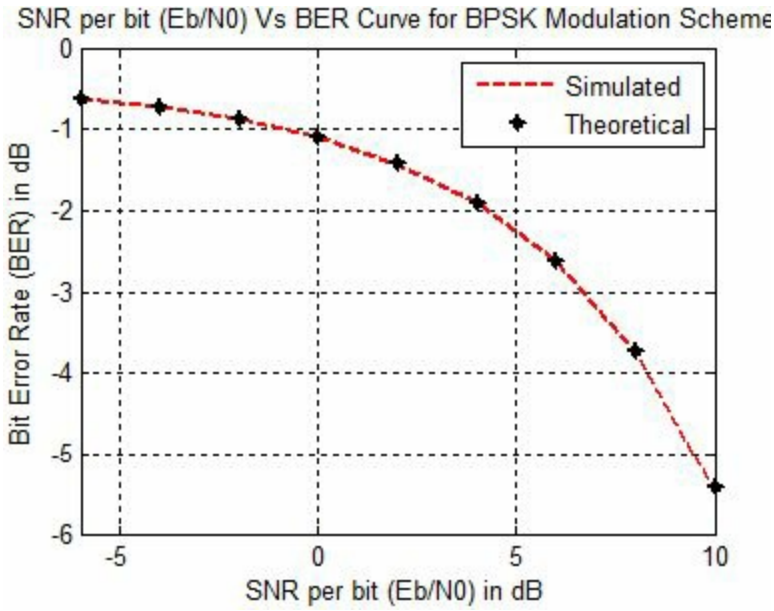
Matlab code:

% Demonstration of Eb/N0 Vs BER for BPSK modulation scheme

```
clear;
clc;
%---------Input Fields----------------------
N=10000000; %Number of input bits
EbN0dB = -6:2:10; % Eb/N0 range in dB for simulation
```

```
%-----------------------------------------

data=randn(1,N)>=0; %Generating a uniformly distributed random 1s and 0s
bpskModulated = 2*data-1; %Mapping 0->-1 and 1->1
M=2; %Number of Constellation points M=2^k for BPSK k=1
Rm=log2(M); %Rm=log2(M) for BPSK M=2
Rc=1; %Rc = code rate for a coded system. Since no coding is used Rc=1

BER = zeros(1,length(EbN0dB)); %Place holder for BER values for each Eb/N0
index=1;

for k=EbN0dB,
    %---------------------------------------
    %Channel Noise for various Eb/N0
    %---------------------------------------
    %Adding noise with variance according to the required Eb/N0
    EbN0 = 10.^(k/10); %Converting Eb/N0 dB value to linear scale
    noiseSigma = sqrt(1./(2*Rm*Rc*EbN0)); %Standard deviation for AWGN Noise
    noise = noiseSigma*randn(1,length(bpskModulated));
    received = bpskModulated + noise;
    %---------------------------------------
    %Threshold Detector
    estimatedBits=(received>=0);
    %---------------------------------------
    %Bit Error rate Calculation
    BER(index) = sum(xor(data,estimatedBits))/length(data);
    index=index+1;
end

%Plot commands follows
plotHandle=plot(EbN0dB,log10(BER),'r--');
set(plotHandle,'LineWidth',1.5);
title('SNR per bit (Eb/N0) Vs BER Curve for BPSK Modulation Scheme');
xlabel('SNR per bit (Eb/N0) in dB');
ylabel('Bit Error Rate (BER) in dB');
grid on;
hold on;

theoreticalBER = 0.5*erfc(sqrt(10.^(EbN0dB/10)));
plotHandle=plot(EbN0dB,log10(theoreticalBER),'k*');
set(plotHandle,'LineWidth',1.5);
legend('Simulated','Theoretical');
grid on;
```

Simulation Result:



******

Back to Table of Contents

6.3 E_b/N_0 vs. BER for BPSK over Rayleigh Channel

The phenomenon of Rayleigh Flat fading and its simulation using Young's model and Clarke's model were discussed in sections 5.7 and 5.8 respectively. The performance (E_b/N_O Vs BER) of BPSK modulation (with coherent detection) over Rayleigh Fading channel and its comparison over AWGN channel is discussed in this section.

We first investigate the non-coherent detection of BPSK over Rayleigh Fading channel and then we move on to the coherent detection. For both the cases, we consider a simple flat fading Rayleigh channel (modeled as a – single tap filter – with complex impulse response – h). The channel also adds AWGN noise to the signal samples after it suffers from Rayleigh Fading.

The received signal y can be represented as

$$y = hx + n$$

Here, n is the noise contributed by AWGN which is Gaussian distributed with zero mean and unit variance and h is the complex channel amplitude scaling factor that follows Rayleigh distribution. (For a simple AWGN channel without Rayleigh Fading the received signal is represented as y=x+n). The one tap complex Rayleigh fading process is generated from two Gaussian random variables with the variance of 0.5 each

Non-Coherent Detection:

In non-coherent detection, prior knowledge of the channel impulse response is not known at the receiver. Consider the BPSK signaling scheme with 'x=+/- a' being transmitted over such a channel as described above. This signaling scheme fails completely (in non-coherent detection scheme), even in the absence of noise, since the phase of the received signal y is uniformly distributed between 0

and 2π regardless of whether x[m]=+a or x[m]=-a is transmitted. So the non-coherent detection of the BPSK signaling is not a suitable method of detection especially in a Fading environment.

Coherent Detection:

In coherent detection, the receiver has sufficient knowledge about the channel impulse response. Techniques like pilot transmissions are used to estimate the channel impulse response or the channel amplitude scaling factor at the receiver, before the actual data transmission could begin. Let us consider that the channel amplitude scaling factor ('h') estimate at receiver is known and is perfect & accurate. The transmitted symbols ('x') can be obtained from the received signal ('y') by the process of equalization as given below.

$$\hat{y} = \frac{y}{h} = \frac{hx + n}{h} = x + z$$

Here, z is still an AWGN noise except for the scaling factor 1/h. Now the detection of x can be performed in a manner similar to the detection in AWGN channels.

The input binary bits ('d') to the BPSK modulation system are detected as

$$r = real(\hat{y}) = real(x + z)$$
$$\hat{d} = 1, \quad if\ r > 0$$
$$\hat{d} = 0, \quad if\ r < 0$$

Theoretical BER:

The theoretical BER for BPSK modulation scheme over Rayleigh fading channel (with AWGN noise) is given by

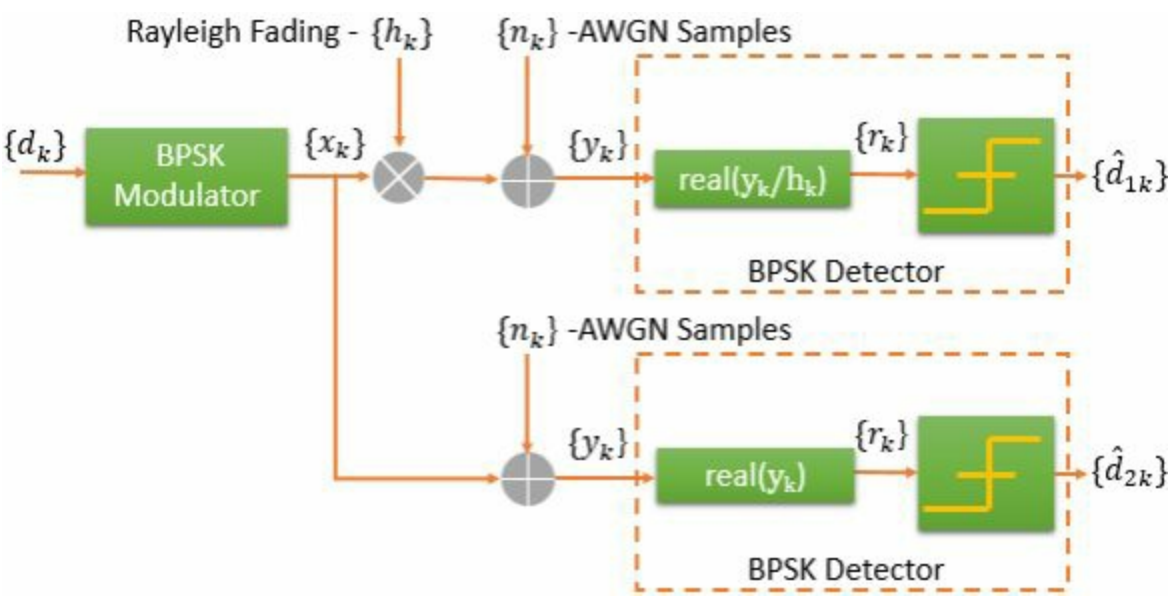
$$P_b = \frac{1}{2}\left(1 - \sqrt{\frac{E_b/N_0}{1 + E_b/N_0}}\right)$$

The theoretical BER for BPSK modulation scheme over an AWGN channel is given here for comparison

$$P_b = \frac{1}{2} erfc\left(\sqrt{E_b/N_0}\right)$$

Simulation Model:

The following model is used for the simulation of BPSK over Rayleigh Fading channel and its comparison with AWGN channel



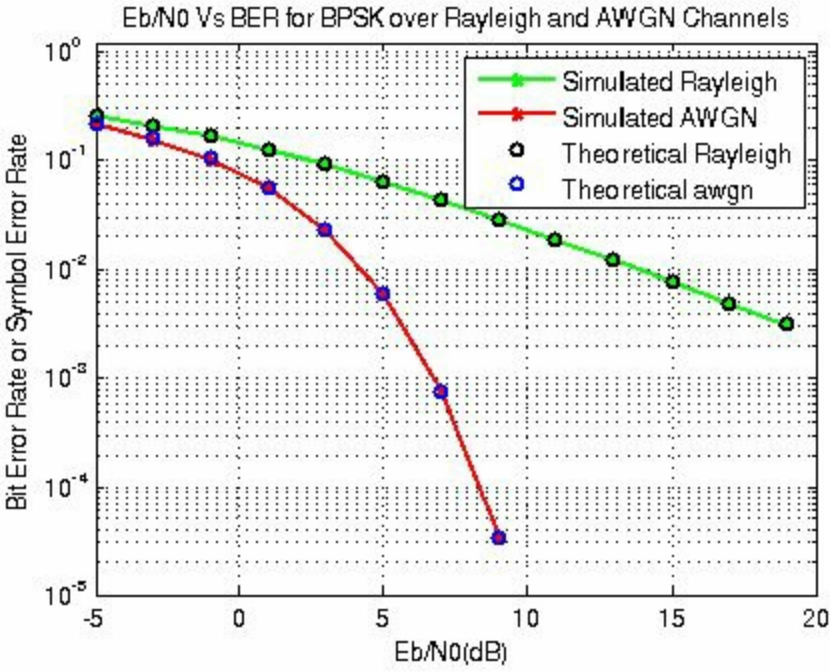
Matlab Code:

```
% Eb/N0 Vs BER for BPSK Modulation over Rayleigh Channel and AWGN

clc;clear;
N=10^6; %Number of BPSK symbols to transmit
d=rand(1,N)>0.5; %binary data
x=2*d-1; %BPSK symbols 0->-1, 1->1
EbN0dB=-5:2:20; %Range of Eb/N0 values

simBER_rayleigh=zeros(1,length(EbN0dB));
simBER_awgn=zeros(1,length(EbN0dB));

for i=1:length(EbN0dB),
    noise=1/sqrt(2)*(randn(1,N)+1i*randn(1,N)); %AWGN noise with mean=0 var=1
    h=1/sqrt(2)*(randn(1,N)+1i*randn(1,N)); %Rayleigh Flat Fading factor- single tap
    n = noise*10^(-EbN0dB(i)/20); %Scaling the noise for required Eb/N0
    y_awgn=x+n; %received signal through AWGN channel
    y_rayleigh=h.*x+n; %received signal through Rayleigh channel
    %Coherent Receiver for Rayleigh Channel
    y_rayleigh_cap=y_rayleigh./h; %Assuming that h is known at the signal accurately
    r_rayleigh=real(y_rayleigh_cap)>0; %received symbols = 1 is real part > 0 or else it is 0

    %Receiver for AWGN channel
    r_awgn=real(y_awgn)>0;
    simBER_rayleigh(i)=sum(xor(d,r_rayleigh));
    simBER_awgn(i)=sum(xor(d,r_awgn));
end

simBER_rayleigh=simBER_rayleigh/N;
simBER_awgn=simBER_awgn/N;

%Theoretical BER;
```

```
EbN0=10.^(EbN0dB/10); %Eb/N0 in Linear Scale
theoretical_rayleigh=0.5*(1-sqrt(EbN0./(1+EbN0)));
theoretical_awgn=0.5*erfc(sqrt(EbN0));

semilogy(EbN0dB,simBER_rayleigh,'g*-','LineWidth',2);hold on;
semilogy(EbN0dB,simBER_awgn,'r*-','LineWidth',2);hold on;
semilogy(EbN0dB,theoretical_rayleigh,'ko','LineWidth',2); hold on
semilogy(EbN0dB,theoretical_awgn,'bo','LineWidth',2);grid on;
axis([-5 20 10^-5 1.2]);
legend('Simulated Rayleigh','Simulated AWGN','Theoretical Rayleigh','Theoretical awgn');
title('Eb/N0 Vs BER for BPSK over Rayleigh and AWGN Channels');
xlabel('Eb/N0(dB)');
ylabel('Bit Error Rate or Symbol Error Rate');
```

Simulation Results:

The simulated and theoretical performance curves (E_b/N_O vs. BER) for BPSK modulation over Rayleigh Fading channel and the AWGN is given below.



******

Back to Table of Contents

6.4 Eb/N0 Vs BER for BPSK over Rician Fading Channel

In one of the previous sections, simulation of BPSK over Rayleigh Fading channel was discussed. This article deals with simulation of another important fading type: Rician Fading.

Rayleigh Fading and Rayleigh Distribution:

Rayleigh Fading model is used to simulate environments that have multiple scatters and Non Line Of

Sight (NLOS) path. If there are sufficient multiple scatters in the environment, all the reflected signals that appear at the receiver front end becomes uncorrelated in amplitude (this brings mean =0) and phase evenly distributed between 0 to 2π. Eventually the in-phase (I) and quadrature (Q) components become Gaussian distributed (by the virtue of central limit theorem) and their envelope $Z = \sqrt{I^2 + Q^2}$ becomes Rayleigh distributed. Note that distribution I^2 is called Chi-Square distribution. If I^2 has mean=0, then it is called central-Chi-square distribution and if mean $\neq$ 0 it is called non-central-Chi-square distribution

Rician Fading and Rician Distribution:

Rician Fading model is used to simulate environments that produce multipath components in addition to a dominant Line Of Sight (LOS) component. The LOS component is called "specular" component and the multipath components are called "random or scatter" components. The amplitude distribution of the specular component will have non-zero mean, whereas, the random component is considered to have zero-mean.

Consider two Gaussian random variables X and Y. Here, the random variable X models the specular component (LOS) and Y models the random/scatter component (NLOS). By definition, X has non-zero mean (m_1=s), Y has zero mean (m_2=0) and both have equal variance σ^2. Then the transformation $Z - \sqrt{X^2 + Y^2}$ is Rician Distributed (Refer Chapter 5.9 on - generating Rician Distribution).

Since, the two variables X and Y have different "means", a non-centrality parameter (indicating the non-central mean) is used for simulation.

$$s = \sqrt{m_1^2 + m_2^2} = \sqrt{s^2 + 0}$$

The non-centrality parameter (the imbalance in the means) is caused by the presence of dominant path in a Rician Fading environment. Due to this, the Rician K factor - representing the ratio of power of Line-Of-Sight (LOS) (or dominant multipath component) and the power of Non-Line-Of-Sight (NLOS) (or the remaining multipath components) is defined in such scenario.

$$\kappa = \frac{Power\ of\ LOS\ component}{Power\ of\ NLOS\ components}$$

Statistically, this can be represented as the power in the faded envelope that has been produced by the means of X and Y.

$$\kappa = \frac{m_1^2 + m_2^2}{2\sigma^2} = \frac{s^2}{2\sigma^2} \quad -(1)$$

The envelope (R) follows Rician distribution, whose PDF is given by

$$f_R(r) = \frac{r}{\sigma^2} \exp\left(-\frac{r^2 + s^2}{2\sigma^2}\right) I_0\left(\frac{rs}{\sigma^2}\right)$$

Where, $I_0(x)$ is the modified zero-th order Bessel function of the first kind.

It can be immediately ascertained that Rayleigh Fading is related to central Chi Square distribution (due to zero mean) and the Rician Fading is related to non-central Chi Square distribution (due to non-zero mean).

Performance simulation of BPSK over Rician Fading Channel with AWGN noise:

A Rician fading channel is described using two parameters:
1) Rician K factor
2) The total power of direct (LOS) path & the scattered paths ($s^2+2\ \sigma^2$).

In the following simulation, the above two parameters are used as inputs.For the following simulation, the Rician K factor and the total power of the dominant & scatter components are given. Therefore, it is imperative that we must formulate the means and variances of the undelying Gaussian distribution from the given Rician K factor.

Let us define s and σ, such that they satisfy the equation (1) above

$$s = \frac{\kappa}{\kappa + 1}.Total\ Power \qquad (2)$$

$$\sigma = \frac{Total\ Power}{\sqrt{2(\kappa + 1)}} \qquad -(3)$$

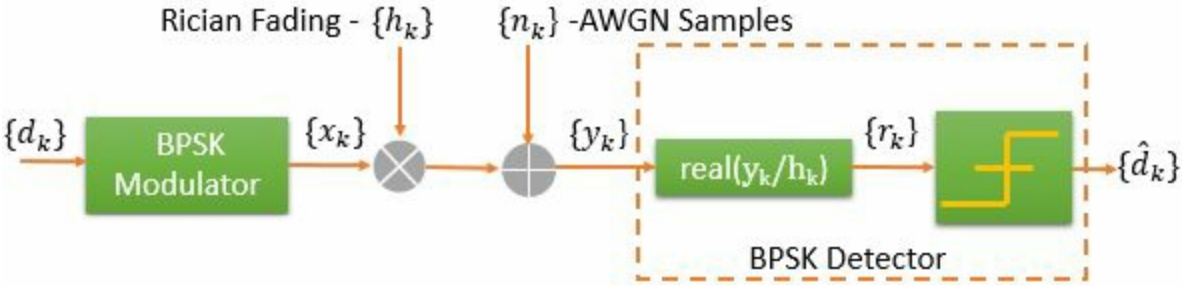
To simulate a Rician Fading channel, mean and standard deviation of the underlying Gaussian RVs have to be calculated with the given Rician K factor and the total power of the components (LOS+NLOS). For simulation purposes, one of the random variable X is generated with mean equal to non-centrality parameter (equation 2) and the other random variable Y is generated with zero mean. Both random variables have same standard deviation (equation 3).

As discussed above, in Rician Fading, the specular component (X) has to be a Gaussian random variable with mean=s and standard deviation = σ, but the scatter component (Y) has to be generated with mean=0 and standard deviation = σ.

In Matlab "randn" function generates Gaussian random numbers with mean = 0 and standard deviation = 1. To generate X component with mean=s and standard deviation = σ, the output of randn has to be multiplied with σ and added with s. To generate Y component with sigma= σ, the output of the randn function has to be multiplied with σ. The rest of the simulation code is similar to that of BPSK over Rayleigh/AWGN channel simulation.

Simulation Model:

In the following simulation model, the total power of dominant and scattered components is assumed to be unity.The results are plotted for various Rician K factors.

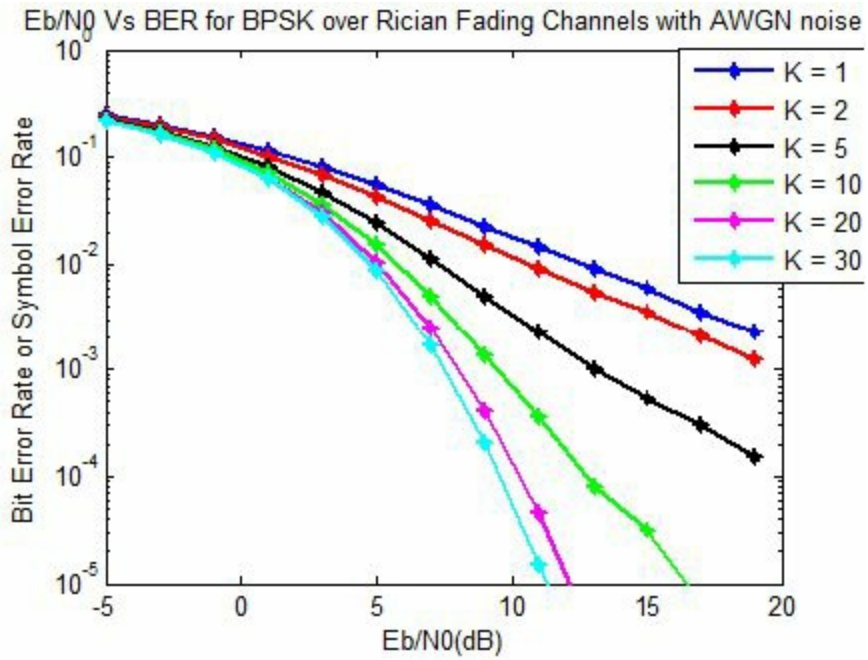


Matlab Code:

%Eb/N0 Vs BER for BPSK over Rician Fading Channel with AWGN noise

```
clc;clear all;
%--------Inputs----------------------------------------------
N=10^6; %Number of data samples to send across the Rician Channel
EbN0dB=-5:2:20; %Eb/N0 in dB overwhich the performance has to be simulated
totPower=1; %Total power of LOS path & scattered paths
K=[1 2 5 10 20 30]; %A list of Ricial K factors to simulate
%-------------------------------------------------------------

d=rand(1,N)>0.5; %data generation
x=2*d-1; %BPSK modulation

simBER_ricean=zeros(1,length(EbN0dB));

plotStyle={'b*-','r*-','k*-','g*-','m*-','c*-'};
for index =1:length(K)
    k=K(index);
    %Derive non-centrality parameter and sigma for the underlying
    %Gaussian RVs to generate the Rician Envelope
    s=sqrt(k/(k+1)*totPower); %Non-Centrality Parameter
    sigma=totPower/sqrt(2*(k+1));

    for i=1:length(EbN0dB)
        noise=1/sqrt(2)*(randn(1,N)+1i*randn(1,N)); %AWGN noise with mean=0 var=1
        h=((sigma*randn(1,N)+s)+1i*(randn(1,N)*sigma+0)); %Rician Fading - single tap
        n = noise*10^(-EbN0dB(i)/20); %Scaling the noise for required Eb/N0
        y_ricean=h.*x+n; %received signal through Rician channel
        %Coherent Receiver for Rician Channel
        y_ricean_cap=y_ricean./h; %Assuming that h is known at the signal accurately
        r_ricean=real(y_ricean_cap)>0; %received symbols = 1 is real part > 0 or else it is 0
        %Receiver for AWGN channel
        simBER_ricean(i)=sum(xor(d,r_ricean));
    end
    simBER_ricean=simBER_ricean/N;
    %Simulated BER;
    EbN0=10.^(EbN0dB/10); %Eb/N0 in Linear Scale
    semilogy(EbN0dB,simBER_ricean,plotStyle{index},'LineWidth',2);hold on
    legendInfo{index} = ['K = ' num2str(K(index))];
end
axis([-5 20 10^-5 10^0]);
legend(legendInfo);
title('Eb/N0 Vs BER for BPSK over Rician Fading Channels with AWGN noise');
xlabel('Eb/N0(dB)');
ylabel('Bit Error Rate or Symbol Error Rate');
```

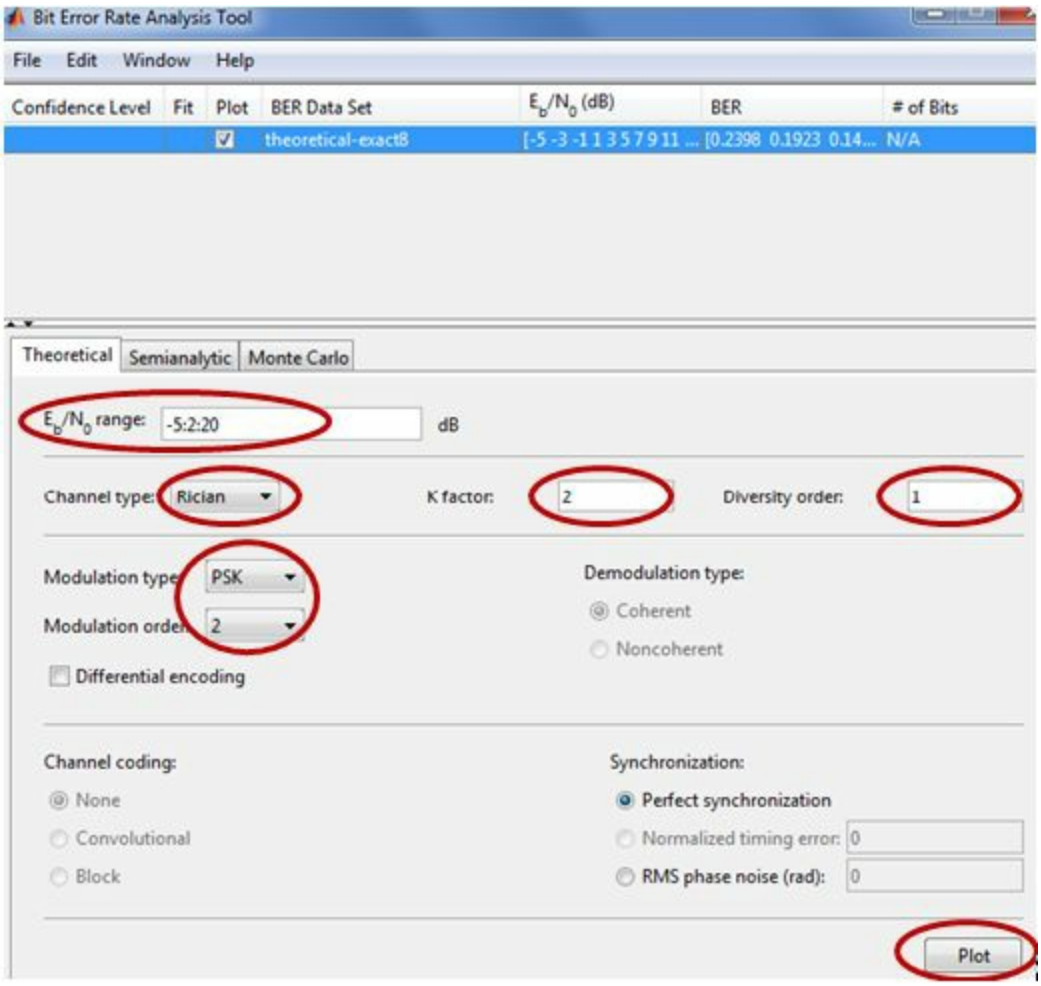
Simulation Result:



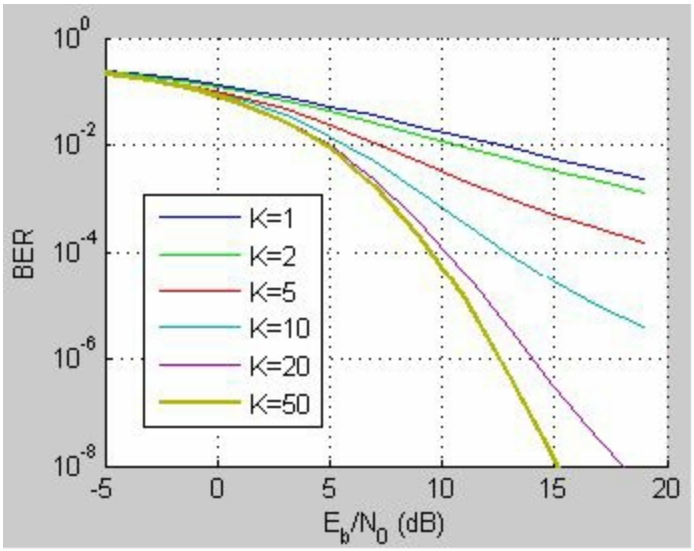
Verifying the theoretical BER using Matlab's BERTOOL:

Theoretical BER for BPSK over Rician Fading Channel with AWGN noise is complicated. Nevertheless you can check [Lindsey1964] to know more on the derivation.

Alternatively, the theoretical results can be cross-checked with Matlab's BERTOOL. Invoke the tool by typing "bertool" in Matlab's command prompt.The appropriate settings for getting the theoretical result for performance of BPSK over Rician Fading channel (with K factor=2) is shown in the next figure.



The theoretical performance results for several K factors given by BERTOOL are shown below.



The data set can be exported to workspace and plotted against the simulated curves for verification.

******

Back to Table of Contents

6.5 QPSK Modulation and Demodulation

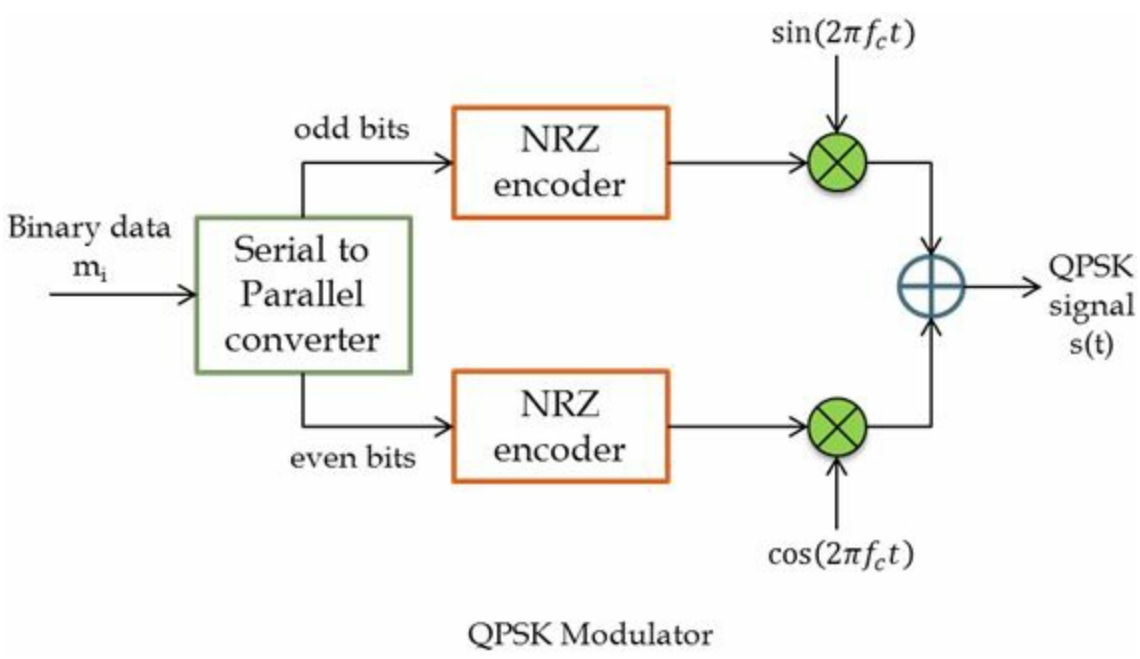
QPSK Modulation:

In Quadrature Phase Shift Keying (QPSK) two sinusoids (sin and cos functions) are taken as basis functions for modulation. Modulation is achieved by varying the phase of the basis functions depending on the message symbols. In QPSK, modulation is symbol based, where one symbol contains 2 bits. The following equation outlines QPSK modulation technique.

$$s_i(t) = \sqrt{\frac{2E_s}{T}} cos\left[2\pi f_c t + (2n-1)\frac{\pi}{4}\right] \quad ; n = 1,2,3,4$$

This gives four possible phases: π/4, 3π/4, 5π/4, 7π/4. The constellation diagram of QPSK will show the constellation points lying on both x and y axes. This means that the QPSK modulated signal will have an in-phase component (I) and also a quadrature component (Q).

A QPSK modulator can be implemented as follows. A de-multiplexer (or serial to parallel converter) is used to separate odd and even bits from the generated information bits. Each of the odd bits (quadrature arm) and even bits (in-phase arm) are converted to NRZ format in a parallel manner. The signal on the in-phase arm is multiplied by cosine component and the signal on the quadrature arm is multiplied by sine component. QPSK modulated signal is obtained by adding the signal from both in-phase and quadrature arm.

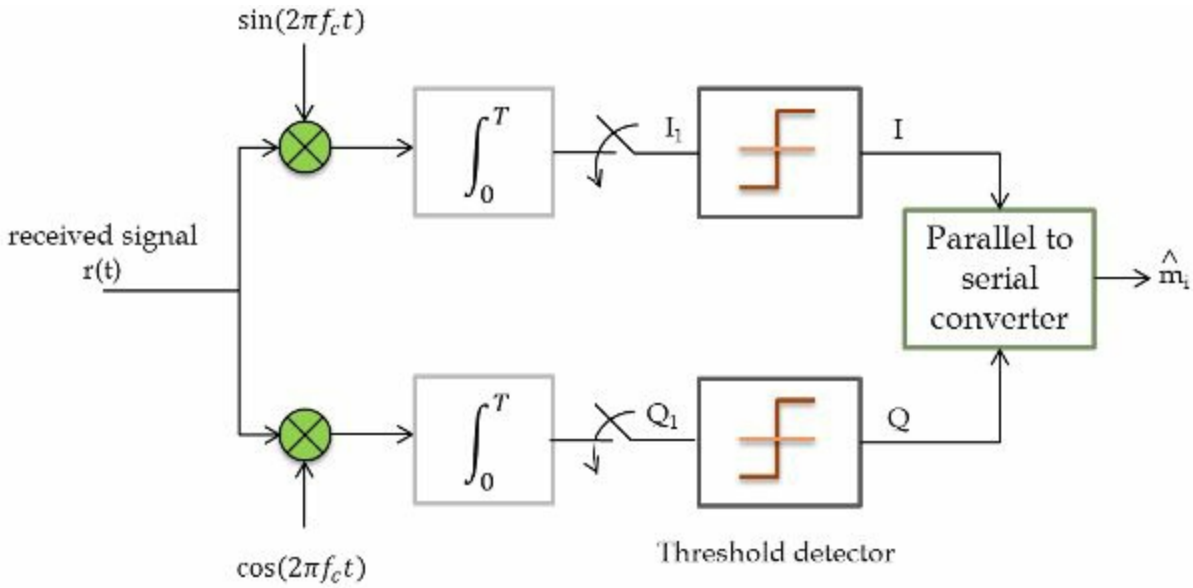


QPSK Modulator

QPSK Demodulation:

For QPSK demodulator, a coherent demodulator is taken as an example. In coherent detection technique, the knowledge of the carrier frequency and phase must be known to the receiver. This can be achieved by using a PLL (phase lock loop) at the receiver. A PLL essentially locks to the incoming carrier frequency and tracks the variations in frequency and phase. For the following simulation, a PLL is not used but instead we simply assume that the receiver is in perfect synchronization. For demonstration purposes we simply assume that the carrier phase recovery is done and simply use the generated reference frequencies at the receiver ($\cos(2\pi f_C t)$and ($\sin(2\pi f_C t)$).

In the demodulator the received signal is multiplied by a reference frequency generator ($\cos(2\pi f_C t)$) and ($\sin(2\pi f_C t)$) on separate arms (in-phase and quadrature arms). The multiplied output on each arm is integrated over one bit period using an integrator. A threshold detector makes a decision on each integrated bit based on a threshold. Finally the bits on the in-phase arm (even bits) and on the quadrature arm (odd bits) are remapped to form the detected information stream.



QPSK Demodulator

Matlab Code:

File1: QPSK.m

```
% Demonstration of QPSK Modulation and Demodulation

clear; %clear all stored variables
N=100; %number of data bits
noiseVariance = 0.1; %Noise variance of AWGN channel
Rb=1e3; %bit rate
amplitude=1; % Amplitude of NRZ data
data=randn(1,N)>=0; %Generate uniformly distributed random data
oddBits = data(1:2:end);
evenBits= data(2:2:end);
[evenTime,evenNrzData,Fs]=NRZ_Encoder(evenBits,Rb,amplitude,'Polar');
[oddTime,oddNrzData]=NRZ_Encoder(oddBits,Rb,amplitude,'Polar');
Fc=2*Rb;

inPhaseOsc = 1/sqrt(2)*cos(2*pi*Fc*evenTime);
quadPhaseOsc = 1/sqrt(2)*sin(2*pi*Fc*oddTime);
qpskModulated = oddNrzData.*quadPhaseOsc + evenNrzData.*inPhaseOsc;
Tb=1/Rb;

subplot(3,2,1);
stem(data);
xlabel('Samples');
ylabel('Amplitude');
title('Input Binary Data');
axis([0,N,-0.5,1.5]);
subplot(3,2,3);
plotHandle=plot(qpskModulated);
xlabel('Samples');
ylabel('Amplitude');
title('QPSK modulated Data');
xlimits = XLIM;
ylimits = YLIM;
axis([xlimits,ylimits(1)-0.5,ylimits(2)+0.5]) ;
grid on;

%----------------------------------------
%Adding Channel Noise
%----------------------------------------
noise = sqrt(noiseVariance)*randn(1,length(qpskModulated));
received = qpskModulated + noise;

subplot(3,2,5);
plot(received);
xlabel('Time');
```

```
ylabel('Amplitude');
title('QPSK Modulated Data with AWGN noise');
%----------------------------------------
%QPSK Receiver
%----------------------------------------
%Multiplying the received signal with reference Oscillator
iSignal = received.*inPhaseOsc;
qSignal = received.*quadPhaseOsc;

%Integrator
integrationBase = 0:1/Fs:Tb-1/Fs;

for i = 0:(length(iSignal)/(Tb*Fs))-1,
    inPhaseComponent(i+1)=trapz(integrationBase,iSignal(int32(i*Tb*Fs+1):int32((i+1)*Tb*Fs)));
end

for i = 0:(length(qSignal)/(Tb*Fs))-1,
    quadraturePhaseComponent(i+1)=trapz(integrationBase,qSignal(int32(i*Tb*Fs+1):int32((i+1)*Tb
end

%Threshold Comparator
estimatedInphaseBits=(inPhaseComponent>=0);
estimatedQuadphaseBits=(quadraturePhaseComponent>=0);

finalOutput=reshape([estimatedQuadphaseBits;estimatedInphaseBits],1,[]);

BER = sum(xor(finalOutput,data))/length(data);

subplot(3,2,2);
stem(finalOutput);
xlabel('Samples');
ylabel('Amplitude');
title('Detected Binary Data after QPSK demodulation');
axis([0,N,-0.5,1.5]);

%Constellation Mapping at transmitter and receiver
%constellation Mapper at Transmitter side
subplot(3,2,4);
plot(evenNrzData,oddNrzData,'ro');
xlabel('Inphase Component');
ylabel('Quadrature Phase component');
title('QPSK Constellation at Transmitter');
axis([-1.5,1.5,-1.5,1.5]);
h=line([0 0],[-1.5 1.5]);
set(h,'Color',[0,0,0])
```

```
h=line([-1.5 1.5],[0 0]);
set(h,'Color',[0,0,0])

%constellation Mapper at receiver side
subplot(3,2,6);
%plot(inPhaseComponent/max(inPhaseComponent),quadraturePhaseComponent/max(quadraturePhase
plot(2*estimatedInphaseBits-1,2*estimatedQuadphaseBits-1,'ro');
xlabel('Inphase Component');
ylabel('Quadrature Phase component');
title(['QPSK Constellation at Receiver when AWGN Noise Variance =',num2str(noiseVariance)]);
axis([-1.5,1.5,-1.5,1.5]);
h=line([0 0],[-1.5 1.5]);
set(h,'Color',[0,0,0]);
h=line([-1.5 1.5],[0 0]);
set(h,'Color',[0,0,0]);
```

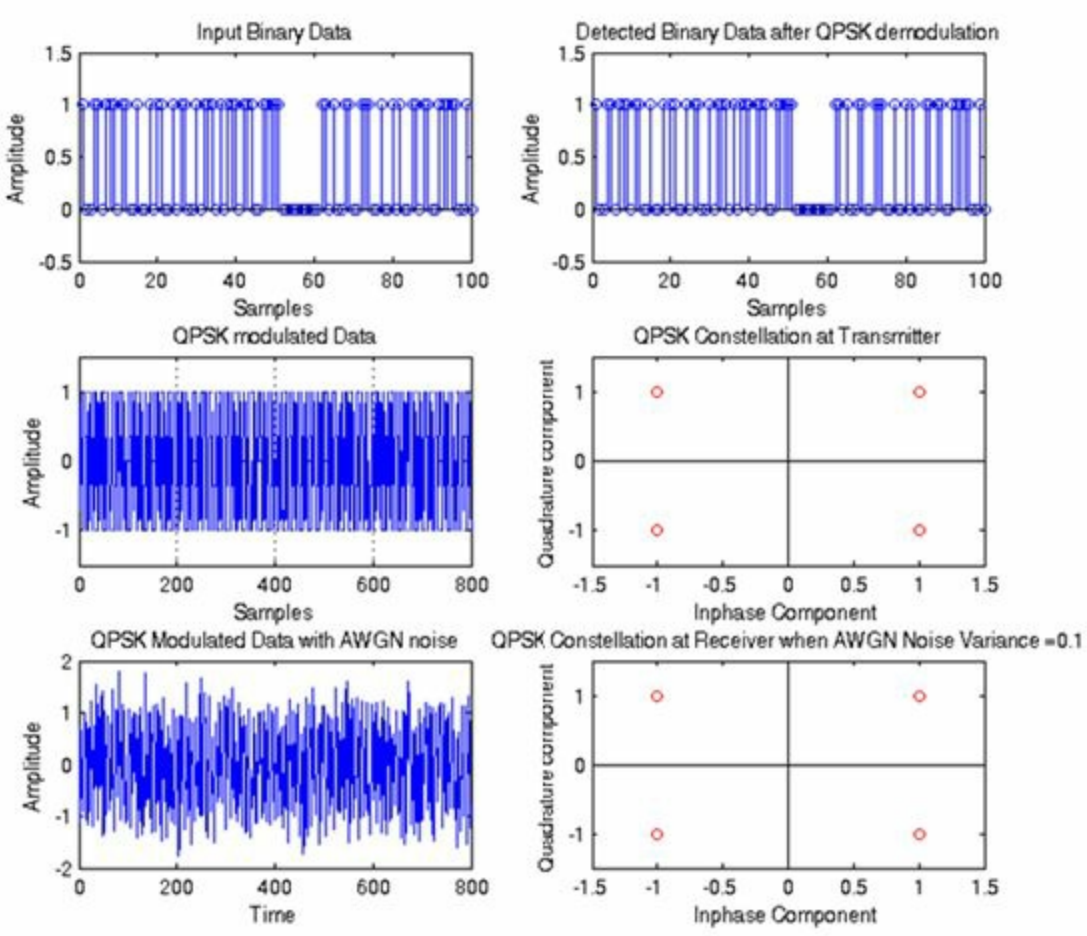
File2: NRZ_Encoder.m

```
% NRZ_Encoder   Line codes encoder
%[time,output,Fs]=NRZ_Encoder(input,Rb,amplitude,style)
%NRZ_Encoder NRZ encoder
%Input a stream of bits and specify bit-Rate, amplitude of the output signal and the style of encoding
%Currently 3 encoding styles are supported namely 'Manchester','Unipolar'and 'Polar'
%Outputs the NRZ stream
function [time,output,Fs]=NRZ_Encoder(input,Rb,amplitude,style)
Fs=10*Rb; %Sampling frequency , oversampling factor= 32
Ts=1/Fs; % Sampling Period
Tb=1/Rb; % Bit period
output=[];
switch lower(style)
case {'manchester'}
for count=1:length(input),
for tempTime=0:Ts:Tb/2-Ts,
output=[output (-1)^(input(count))*amplitude];
end
for tempTime=Tb/2:Ts:Tb-Ts,
output=[output (-1)^(input(count)+1)*amplitude];
end
end
case {'unipolar'}
for count=1:length(input),
for tempTime=0:Ts:Tb-Ts,
output=[output input(count)*amplitude];
end
end
case {'polar'}
```

```
for count=1:length(input),
for tempTime=0:Ts:Tb-Ts,
output=[output amplitude*(-1)^(1+input(count))];
end
end
otherwise,
disp('NRZ_Encoder(input,Rb,amplitude,style)-Unknown method given as "style" argument');
disp('Accepted Styles are "Manchester", "Unipolar" and "Polar"');
end
time=0:Ts:Tb*length(input)-Ts;
```

The output of the Matlab code provides more insight into the QPSK modulation technique. Apart from plotting the modulated and demodulated signal it also shows the constellation at transmitter/receiver. This Matlab code is written without using any of the inbuilt functions from communication toolbox.

Simulation Result:



******

Back to Table of Contents

6.6 BER vs. E_b/N_0 for QPSK modulation over AWGN

In the previous section, we saw how QPSK modulation and demodulation in passband can be done.

This concept is extended further to simulate the performance of baseband QPSK modulation technique over an AWGN.

Transmitter:

For the QPSK modulation, a series of binary input message bits are generated. In QPSK, a symbol contains 2 bits. The generated binary bits are combined in terms of two bits and QPSK symbols are generated. From the constellation of QPSK modulation the symbol '00' is represented by 1, '01' by j (90 degrees phase rotation), '10' by -1 (180 degrees phase rotation) and '11' by -j (270 degrees phase rotation). In another form of QPSK, these phase rotations are offset by 45 degrees. So the effective representation of symbols in this form of QPSK is '00'=1+j (45 degrees), '01'=-1+j (135 degrees), '10' = -1-j (225 degrees) and '11'= 1-j (315 degrees).

Here we are simulating a 45* rotated QPSK system. Once the symbols are mapped, the power of the QPSK modulated signal needs to be normalized by 1/sqrt(2).

AWGN channel:

For QPSK modulation the channel can be modeled as

$$y = ax + n$$

Where, y is the received signal at the input of the QPSK receiver, x is the complex modulated signal transmitted through the channel, 'a' is the channel amplitude scaling factor for the transmitted signal usually set to 1, 'n' is the Additive Gaussian White Noise random variable with zero mean and variance σ^2. For AWGN the noise variance in terms of noise power spectral density (N_0) is given by

$$\sigma^2 = \frac{N_0}{2}$$

For M-PSK modulation schemes including BPSK, the symbol energy is given by

$$E_S = R_m R_c E_b$$

Where, E_S =Symbol energy per modulated bit (x), $R_m = \log_2(M)$, (for BPSK M=2, QPSK M=4, 16 QAM M=16 etc..,). R_C is the code rate of the system if a coding scheme is used. In this case since no coding scheme is used Rc = 1. E_b is the Energy per information bit.

Assuming E_S=1 for QPSK (Symbol energy normalized to 1) E_b/N_0 can be represented as (using above equations),

$$\frac{E_b}{N_0} = \frac{E_S}{R_m R_c N_0} = \frac{E_S}{R_m R_c 2\sigma^2} = \frac{1}{2R_m R_c \sigma^2}$$

From the above equation the noise variance for the given E_b/N_0 can be calculated as

$$\sigma^2 = \left[2R_m R_c \frac{E_b}{N_0}\right]^{-1}$$

For the channel model "randn" function in Matlab is used to generate the noise term. This function generates noise with unit variance and zero mean. In order to generate a noise with standard deviation - σ for the given E_b/N_0 ratio, use the above equation, find σ, multiply the "randn" generated noise with this σ, add this final noise term with the transmitted signal to get the received signal. For a $\pi/4$ rotated

QPSK system, since the modulated signal is in complex form (due to sine and cosine basis functions), the noise should also be in complex form.

Receiver:

QPSK receiver employs two threshold detectors that detect real (in phase arm) and imaginary parts (quadrature arm). The detected signals are sent through a parallel to serial converter (implemented by "reshape" function in MATLAB).

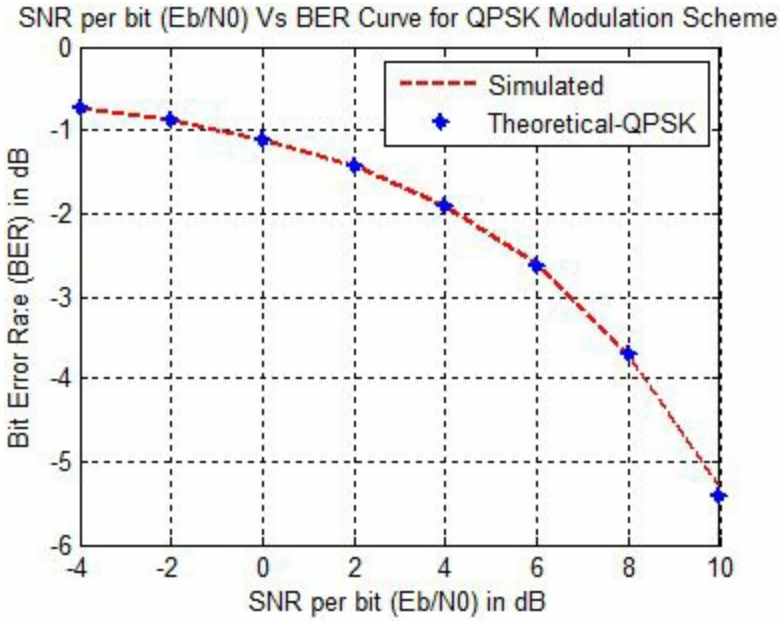
The E_b/N_0 Vs BER Curve for QPSK and BPSK are identical. This is because; QPSK actually consists of two orthogonal BPSK systems. Since the two individual BPSK systems are orthogonal, they don't interfere with each other. That is why the BER curves for QPSK and BPSK are identical.

Matlab Code:

```
% Demonstration of Eb/N0 Vs BER for QPSK modulation scheme

clear;
clc;
%---------Input Fields-----------------------
N=1000000;%Number of input bits
EbN0dB = -4:2:10; % Eb/N0 range in dB for simulation
%--------------------------------------------

data=randn(1,N)>=0; %Generating a uniformly distributed random 1s and 0s
oddData = data(1:2:end);
evenData = data(2:2:end);
qpskModulated = sqrt(1/2)*(1i*(2*oddData-1)+(2*evenData-1)); %QPSK Mapping

M=4; %Number of Constellation points M=2^k for QPSK k=2
Rm=log2(M); %Rm=log2(M) for QPSK M=4
Rc=1; %Rc = code rate for a coded system. Since no coding is used Rc=1

BER = zeros(1,length(EbN0dB)); %Place holder for BER values for each Eb/N0

index=1;

for i=EbN0dB,
    %---------------------------------------
    %Channel Noise for various Eb/N0
    %---------------------------------------
    %Adding noise with variance according to the required Eb/N0
    EbN0 = 10.^(i/10); %Converting Eb/N0 dB value to linear scale
    noiseSigma = sqrt(1./(2*Rm*Rc*EbN0)); %Standard deviation for AWGN Noise

    %Creating a complex noise for adding with QPSK modulated signal
    %Noise is complex since QPSK is in complex representation
    noise = noiseSigma*(randn(1,length(qpskModulated))+1i*randn(1,length(qpskModulated)));
```

```
    received = qpskModulated + noise;

    %----------------------------------------
    %Threshold Detector
    detected_real = real(received)>=0;
    detected_img = imag(received)>=0;
    estimatedBits=reshape([detected_img;detected_real],1,[]);

    %----------------------------------------
    %Bit Error rate Calculation
    BER(index) = sum(xor(data,estimatedBits))/length(data);
    index=index+1;
end

%Plot commands follows
plotHandle=plot(EbN0dB,log10(BER),'r--');
set(plotHandle,'LineWidth',1.5);
title('SNR per bit (Eb/N0) Vs BER Curve for QPSK Modulation Scheme');
xlabel('SNR per bit (Eb/N0) in dB');
ylabel('Bit Error Rate (BER) in dB');
grid on;
hold on;

theoreticalBER =0.5*erfc(sqrt(10.^(EbN0dB/10)));
plotHandle=plot(EbN0dB,log10(theoreticalBER),'b*');
set(plotHandle,'LineWidth',1.5);
legend('Simulated','Theoretical-QPSK','Theoretical-QPSK');
grid on;
```

Simulation Result:



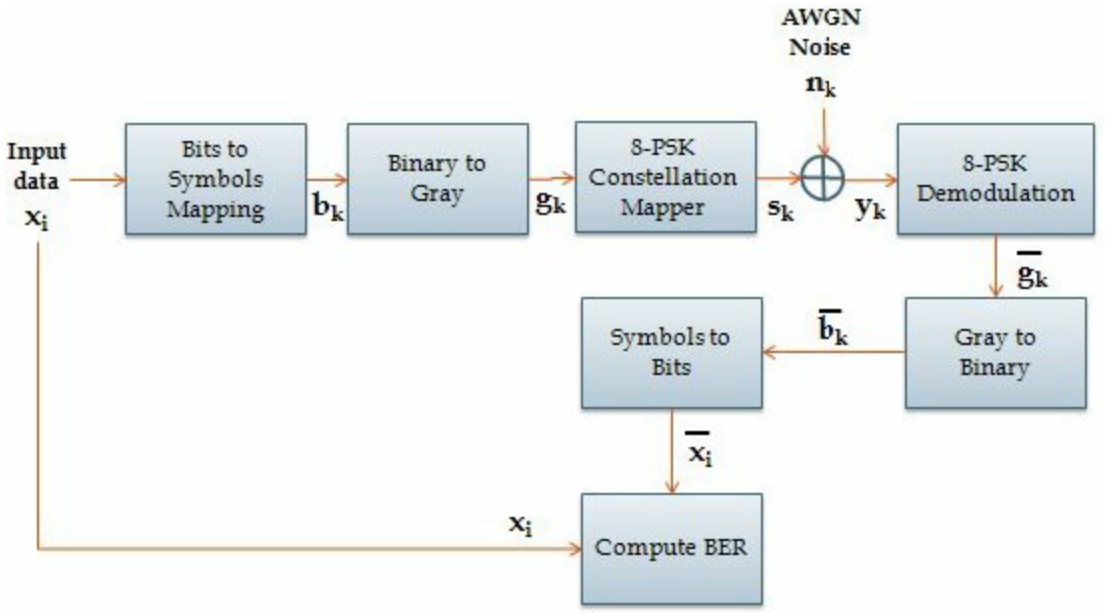
******

Back to Table of Contents

6.7 BER vs. Eb/N0 for 8-PSK Modulation over AWGN

The performance characteristics for QPSK and BER were simulated in the previous sections. BER vs. E_b/N_0 performance curves for 8-PSK is given here. A baseband simulation model for 8-PSK modulation is given below.

Simulation Model:



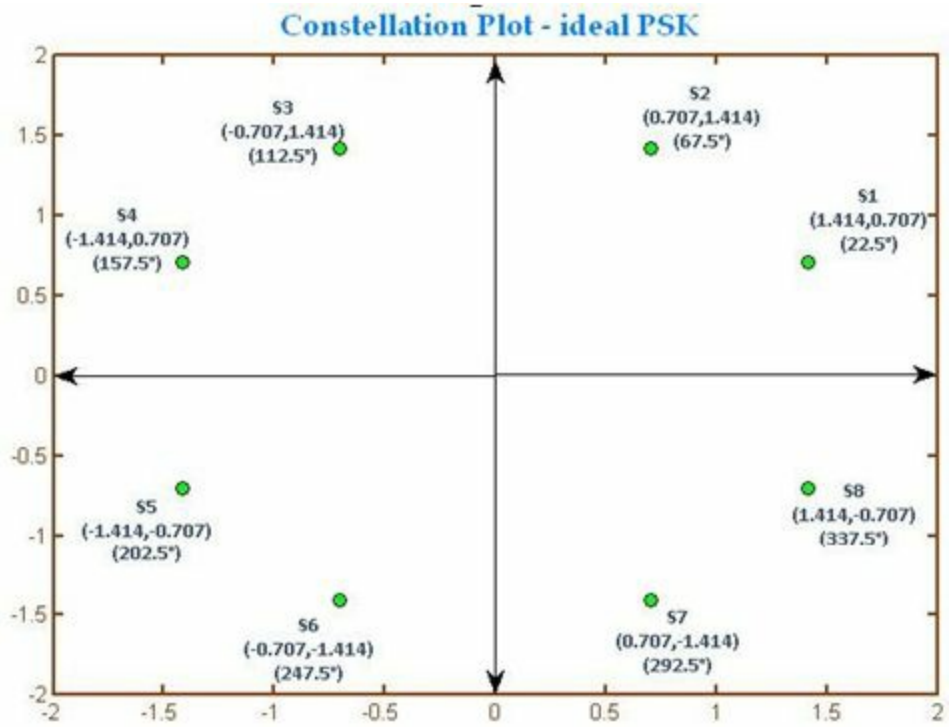
Transmitter:

A complex baseband M-PSK signal is represented by

$$s_n(t) = \sqrt{\frac{2E_s}{T}} cos(\omega t + \theta_n) \quad ,n = 1,2,\dots,M$$

where E_S is the symbol energy of each M-PSK Symbol, T is the bit period, θ_n is the phase shift for each of the symbol. For 8-PSK, M=8.

In 8-PSK modulator, every 3 input data bits are mapped to one complex-valued 8-PSK symbol (whereas in QPSK 2 input bits are mapped to one complex-valued QPSK symbol). Each symbol is gray coded prior to mapping them to the 8-PSK constellation (this is to allow only one bit change in the constellation, thereby avoiding abrupt phase changes in the constellation which may lead to additional errors in the receiver) . In 8PSK there are eight possible symbols that can be transmitted. We will use eight different phases (22.5 °, 67.5 °, 112.5 °, 157.5 °, 202.5 °, 247.5 °, 292.5 °, 337.5 °) to represent each 8-PSK symbol. Each symbol is 45° away from adjacent symbols (see the constellation diagram given in the next figure).



The 8PSK symbols are transmitted over an AWGN channel that adds Gaussian Noise to the transmitted symbol.

Receiver:

The detection process is performed by finding one of the eight constellation symbols that has the smallest Euclidean distance with the received symbol. Some may prefer to use a sector based detection approach in which the received signal constellation is divided into sectors. For 8-PSK, the entire constellation can be equally divided into 8 sectors, with each angle of the sector corresponding to one of the 8 constellation symbols. The detection process can be performed by using the angle of sectors. For example, the angle sector defined by [0, $\pi/4$] corresponds to the 8-PSK symbol S_1. Instead of this approach, this simulation is based on minimum Euclidean Distance. The detector computes the Euclidean Distance of each received symbol against all possible ideal symbols. Then it pin points an ideal symbol which gives the minimum Euclidean Distance.

Euclidean Distance is computed using the following equation

$$D = \sqrt{(y_r - s_r)^2 + (y_i - s_i)^2}$$

Where y=received signal (with noise), s=points on ideal constellation and the subscripts ‘r’ and ‘i’ are for their real and imaginary part respectively.

The detector calculates the Euclidean distance of a received symbol against all possible symbols in the constellation and picks a symbol from the ideal constellation that gives the minimum Euclidean Distance.

Finally, the demodulated symbols are converted back to binary bits and the BER (Bit Error Rate) is calculated.

Matlab Code:

File1: MPSK_8_Simulation.m

```
%8PSK Simulation

clc; clear; close all;
N=100000; %Number of data bits to send over the channel
EbN0dB=-6:2:12;

N=N+rem((3-rem(N,3)),3); %add additional bits to the data to make the length multiple of 3

% one 8-PSK symbol contains 3 binary bits
x=rand(1,N)>=0.5;%Generate random 1's and 0's as data;

%Club 3 bits together and gray code it individually
inputSymBin=reshape(x,3,N/3)';
g=bin2gray(inputSymBin);

%Convert each Gray coded symbols to decimal value this is to ease the process of mapping based on
array index
b=bin2dec(num2str(g,'%-1d'))';

%8-PSK Constellation Mapper
%8-PSK mapping Table
map=[1.414 0.707;0.707 1.414;-0.707 1.414;-1.414 0.707;-1.414 -0.707;-0.707 -1.414;0.707 -
1.414;1.414 -0.707];
s=map(b(:)+1,1)+1i*map(b(:)+1,2);

%Simulation for each Eb/N0 value
M=8; %Number of Constellation points M=2^k for 8-PSK k=3
Rm=log2(M); %Rm=log2(M) for 8-PSK M=8
Rc=1; %Rc = code rate for a coded system. Since no coding is used Rc=1
simulatedBER = zeros(1,length(EbN0dB));
theoreticalBER = zeros(1,length(EbN0dB));
count=1;
figure;
for i=EbN0dB
    %----------------------------------------
    %Channel Noise for various Eb/N0
    %----------------------------------------
    %Adding noise with variance according to the required Eb/N0
    EbN0 = 10.^(i/10); %Converting Eb/N0 dB value to linear scale
    noiseSigma = sqrt(2)*sqrt(1./(2*Rm*Rc*EbN0)); %Standard deviation for AWGN Noise

    %Creating a complex noise for adding with 8-PSK s signal
    %Noise is complex since 8-PSK is in complex representation
    n = noiseSigma*(randn(1,length(s))+1i*randn(1,length(s)))';
```

```
    y = s + n;
    plot(real(y),imag(y),'r*');hold on;
    plot(real(s),imag(s),'ko','MarkerFaceColor','g','MarkerSize',7);hold off;
    title(['Constellation plots - ideal 8-PSK (green) Vs Noisy y signal for EbN0dB =',num2str(i),'
    dB']);
    pause;

    %Demodulation
    %Find the signal points from MAP table using minimum Euclidean distance
    demodSymbols = zeros(1,length(y));
    for j=1:length(y)
        [minVal,minindex]=min(sqrt((real(y(j))-map(:,1)).^2+(imag(y(j))-map(:,2)).^2));
        demodSymbols(j)=minindex-1;
    end
    demodBits=dec2bin(demodSymbols)-'0'; %Dec to binary vector
    xBar=gray2bin(demodBits)'; %gray to binary
    xBar=xBar(:)';
    bitErrors=sum(sum(xor(x,xBar)));
    simulatedBER(count) = log10(bitErrors/N);
    theoreticalBER(count) = log10(1/3*erfc(sqrt(EbN0*3)*sin(pi/8)));
    count=count+1;
end

figure;
plot(EbN0dB,theoreticalBER,'r-*');hold on;
plot(EbN0dB,simulatedBER,'k-o');
title('BER Vs Eb/N0 (dB) for 8-PSK');legend('Theoretical','Simulated');grid on;
xlabel('Eb/N0 dB');
ylabel('BER - Bit Error Rate');
grid on;
```

File 2: gray2bin.m

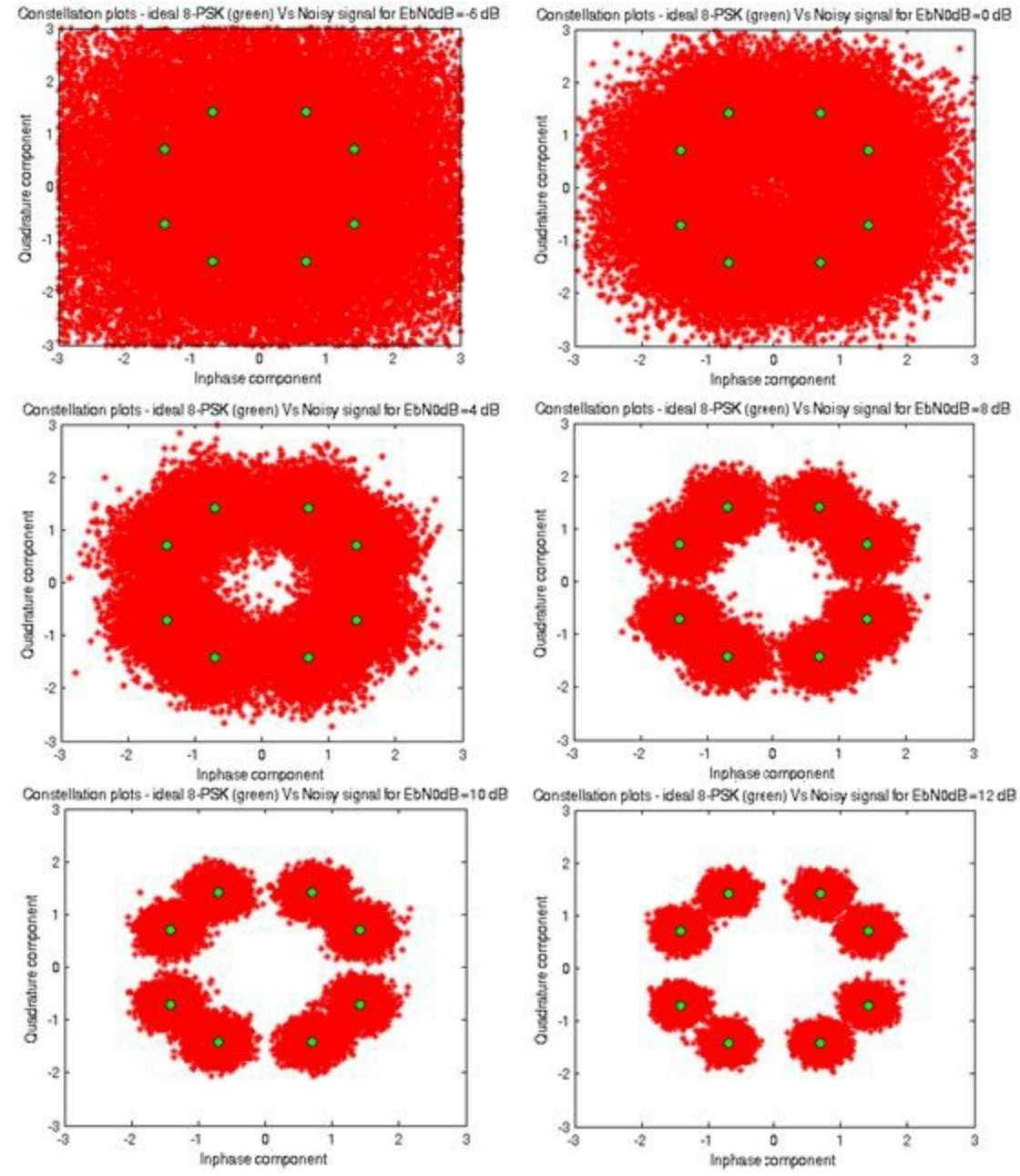
```
function [binaryCoded]=gray2bin(grayInput)
[rows,cols]=size(grayInput);
binaryCoded=zeros(rows,cols);
for i=1:rows
    binaryCoded(i,1)=grayInput(i,1);
    for j=2:cols
        binaryCoded(i,j)=xor(binaryCoded(i,j-1),grayInput(i,j));
    end
end
end
```

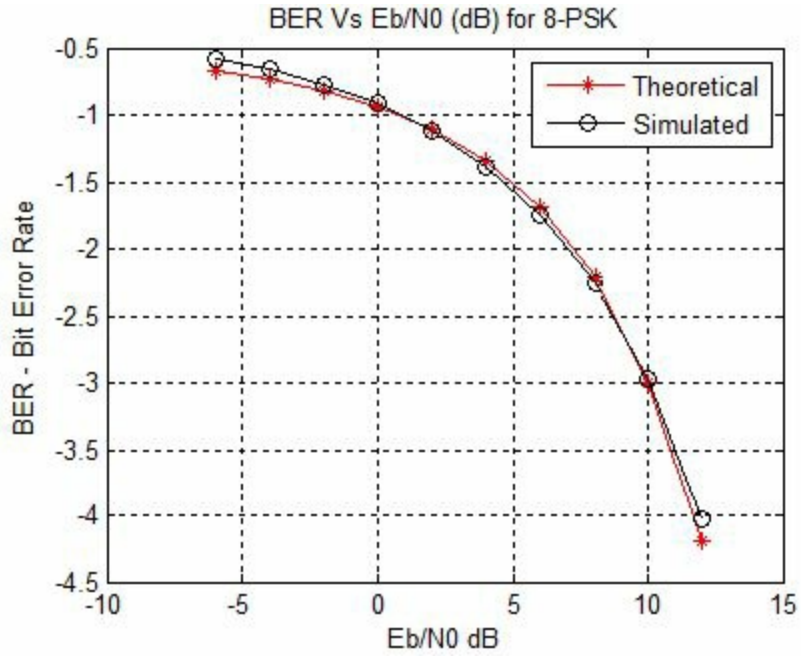
File 3: bin2gray.m

```
function [grayCoded]=bin2gray(binaryInput)
```

```
[rows,cols]=size(binaryInput);
grayCoded=zeros(rows,cols);
for i=1:rows
    grayCoded(i,:)=[binaryInput(i,1) xor(binaryInput(i,2:cols),binaryInput(i,1:cols-1))];
end
end
```

Simulation Results:





******

Back to Table of Contents

6.8 Simulation of M-PSK modulations over AWGN

A generic simulation technique to simulate all M-PSK modulation techniques (for up to M=32) is given here. The given simulation code is very generic, and it plots both simulated and theoretical symbol error rates for all M-PSK modulation techniques (up to M=32).

M-PSK Modulation and simulation methodology:

The general expression for a M-PSK signal set is given by

$$s_i(t) = Vcos\left[2\pi f_c t - \frac{(i-1)2\pi}{M}\right], \quad i = 1,2,\dots,M$$

Here M defines the number of constellation points in the constellation diagram and essentially the M-PSK type. For example M=4 implies 4-PSK or QPSK, M=8 implies 8-PSK. The value of M depends on a parameter 'k' – the number of bits we wish to squeeze into a single M-PSK symbol. For example if we wish to squeeze in 3 bits (k=3) in one transmit symbol, $M = 2^K = 2^3 = 8$. This gives us 8-PSK configuration.

The previous equation can be separated into cosine and sine basis terms as follows

$$s_i(t) = Vcos\left[\frac{(i-1)2\pi}{M}\right]\cos(2\pi f_c t) + Vsin\left[\frac{(i-1)2\pi}{M}\right]\sin(2\pi f_c t)$$

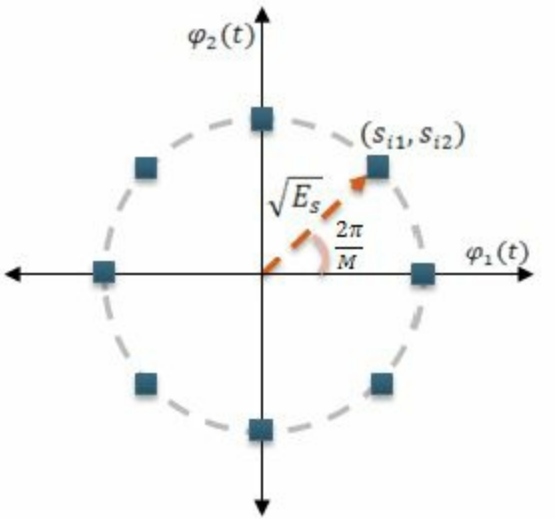
This can be written as,

$$s_i(t) = s_{i1}\phi_1(t) + s_{i2}\phi_2(t), \quad where$$

$$s_{i1} = \sqrt{E_s}\cos\left[\frac{(i-1)2\pi}{M}\right], \quad s_{i2} = \sqrt{E_s}\sin\left[\frac{(i-1)2\pi}{M}\right]$$

$$\phi_1(t) = \frac{V\cos(2\pi f_c t)}{\sqrt{E_s}}, \qquad \phi_2(t) = \frac{V\sin(2\pi f_c t)}{\sqrt{E_s}}$$

Here $\phi_1(t)$and $\phi_2(t)$are the orthonormal basis functions that follows from Gram-Schmidt orthogonalization procedure [Trefethen1997] and s_{i1}, s_{i2} are the coefficients of each signaling point in the M-PSK constellation. E_S is the symbol energy usually normalized to $1/\sqrt{2}$. The constellation points on the M-PSK constellation lie 2π/M radians apart and are placed on a circle of radius $\sqrt{E_s}$. The co-efficient s_{i1} and s_{i2} are termed as in phase (I) and quadrature-phase (Q) components respectively. The ideal constellation diagram for M-PSK contains M equally spaced signaling points that are located at the distance $\sqrt{E_s}$ from the origin. The following figure illustrates the ideal constellation diagram for 8-PSK constellation.



The generated I and Q components are then added with AWGN noise of required variance depending on the required E_S/N_0. In effect, the received signal's constellation is corrupted with noise and the detection is based on comparing the received symbols with the ideal signaling points and making a decision based on the minimum distance.

Finally the simulated and theoretical symbol error rates are computed. The theoretical symbol error rate [Proakis1969] for M-PSK modulation is given by

$$P_s = 2Q\left[\sqrt{2E_s}\sin\left(\frac{\pi}{M}\right)\right] = erfc\left[\sqrt{E_s}\sin\left(\frac{\pi}{M}\right)\right]$$

Matlab Simulation:

File 1: simulateMPSK.m

```
function [simulatedSER, theoreticalSER]= simulateMPSK(M,N,EbN0dB,Rc)
% Generic Function to simulate M-PSK modulation scheme
%Parameter M => defines the MPSK type - M=4 for QPSK, M=8 for 8-PSK M=16 for 16-PSK
M=32 for 32-PSK.
% Parameter N=> Number of symbols to run the simulation
% Parameter EbN0dB => EbN0 in Decibel scale to simulate
% Parameter Rc=> Coding Rate, for an uncoded system Rc=1
```

```
k=log2(M);
EsN0dB = EbN0dB+10*log10(k); % Es/N0 range in dB for simulation , since M-PSK is a symbol modulation we will use EsN0dB for generating proper AWGN noise

%---Generating a uniformly distributed random 1s and 0s--
dataSym=ceil(M.*rand(N,1));

%--- I and Q branch ----------------------
I = 1/sqrt(2)*cos((dataSym-1)/M*2*pi);
Q = 1/sqrt(2)*sin((dataSym-1)/M*2*pi);

%--- Mapping I and Q to one M-PSK symbol---
m_psk = (I+1i*Q); %M-PSK Mapping

%--- Reference Constellation for demodulation and Error rate computation--
s_i=zeros(1,M);s_q=zeros(1,M);
for i=1:1:M
    s_i(i)= 1/sqrt(2)*cos((i-1)/M*2*pi);
    s_q(i)= 1/sqrt(2)*sin((i-1)/M*2*pi);
end

%---Place holder for Symbol Error values for each Es/N0--
simulatedSER = zeros(1,length(EsN0dB));
index=1;

for x=EsN0dB,
    %----------------------------------------
    %Channel Noise for various Eb/N0
    %----------------------------------------
    %Adding noise with variance according to the required Es/N0
    EsN0lin = 10.^(x/10);%Converting Es/N0 dB value to linear scale
    noiseSigma = 1/sqrt(2)*sqrt(1/(2*Rc*EsN0lin));%Standard deviation for AWGN Noise

    %Alternate derivation of noise sigma using Eb/N0
    %EbN0 = EsN0/(Rm*Rc);% Converting Es/N0 to Eb/N0
    %noiseSigma = 1/sqrt(2)*sqrt(1./(2*k*Rc*EbN0)); %Standard deviation for AWGN Noise
    %Creating a complex noise for adding with M-PSK modulated signal
    %Noise is complex since M-PSK is in complex representation
    noise = noiseSigma*(randn(length(m_psk),1)+1i*randn(length(m_psk),1));
    received = m_psk + noise;

    %------------I-Q Branching--------------
    r_i = real(received);
    r_q = imag(received);
```

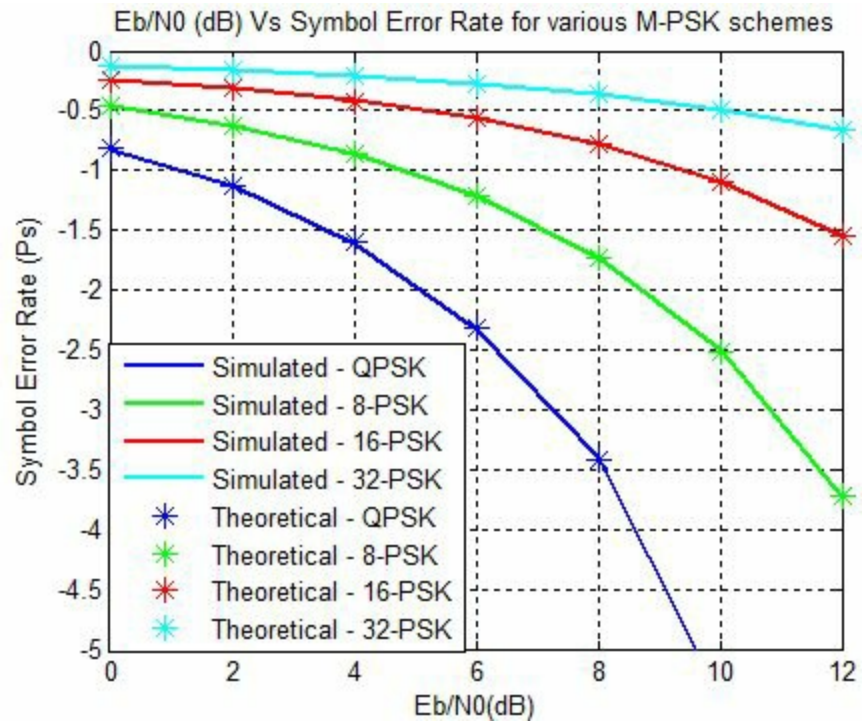
```
    %---Decision Maker-Compute (r_i-s_i)^2+(r_q-s_q)^2 and choose the smallest
    r_i_repmat = repmat(r_i,1,M);
    r_q_repmat = repmat(r_q,1,M);
    distance = zeros(length(r_i),M); %place holder for distance metric
    minDistIndex=zeros(length(r_i),1);
    for j=1:1:length(r_i)
        %---Distance computation - (r_i-s_i)^2+(r_q-s_q)^2 --------------
        distance(j,:) = (r_i_repmat(j,:)-s_i).^2+(r_q_repmat(j,:)-s_q).^2;
        %---capture the index in the array where the minimum distance occurs
        [dummy,minDistIndex(j)]=min(distance(j,:));
    end
    y = minDistIndex; %The index becomes the decoded symbol

    %--------------Symbol Error Rate Calculation-----------------------------
    simulatedSER(index) = sum(y~=dataSym)/N;
    index=index+1;
end

%----- Compute Theoretical Symbol Error Rates --------------------
EbN0lin = 10.^(EbN0dB/10);
theoreticalSER = log10(erfc(sqrt(EbN0lin*k)*sin(pi/M)));
```

File 2: M_PSK_wrapper.m

```
% Demonstration of Eb/N0 Vs SER for all M-PSK modulation scheme

clear;
clc;
%---------Input Fields-----------------------
N=1000000; %Number of input symbols
EbN0dB = 0:2:12; %Define EbN0dB range for simulation
k=2:1:5; % Number of Symbols for each M-PSK modulation
Rc=1; %Rc = code rate for a coded system. Since no coding is used Rc=1
%-----------------------------------------
M=2.^k; %for M-PSK modulation.
simulatedSER = zeros(length(M),length(EbN0dB));
theoreticalSER = zeros(length(M),length(EbN0dB));
plotColor =['b','g','r','c'];
for i=M,
    [simulatedSER(i,:), theoreticalSER(i,:)]= simulateMPSK(i,N,EbN0dB,Rc);
end
%Plot commands
j=1;
for i=M,
    plot(EbN0dB,log10(simulatedSER(i,:)),sprintf('%s-',plotColor(j))); hold on;
    j=j+1;
end
```

```
j=1;
for i=M,
    plot(EbN0dB,theoreticalSER(i,:),sprintf('%s*',plotColor(j))); hold on;
    j=j+1;
end
legend('Simulated - QPSK','Simulated - 8-PSK','Simulated - 16-PSK','Simulated - 32-
PSK','Theoretical - QPSK','Theoretical - 8-PSK','Theoretical - 16-PSK','Theoretical - 32-PSK');
xlabel('Eb/N0(dB)');
ylabel('Symbol Error Rate (Ps)');
title('Eb/N0 (dB) Vs Symbol Error Rate for various M-PSK schemes');
grid on;
```

Simulation Result:



******

Back to Table of Contents

6.9 Symbol Error Rate vs. SNR performance curve simulation for 16-QAM

M-QAM Modulation:

In M-ASK modulation the information symbols (each k=log2 (M) bit wide) are encoded into the amplitude of the sinusoidal carrier. In M-PSK modulation the information is encoded into the phase of the sinusoidal carrier. M-QAM is a generic modulation technique where the information is encoded in both the amplitude and phase of the sinusoidal carrier. It combines both M-ASK and M-PSK modulation techniques.

M-QAM modulation technique is a two dimensional modulation technique and it requires two

orthonormal basis functions

$$\phi_1(t) = \sqrt{\frac{2}{T_s}}\cos(2\pi f_c t) \quad 0 \leq t \leq T_s$$

$$\phi_2(t) = \sqrt{\frac{2}{T_s}}\sin(2\pi f_c t) \quad 0 \leq t \leq T_s$$

The M-QAM modulated signal is represented as

$$s_i(t) = V_{I,i}\sqrt{\frac{2}{T_s}}cos(2\pi f_c t) + V_{Q,i}\sqrt{\frac{2}{T_s}}sin(2\pi f_c t) \quad 0 \leq t \leq T_s \;\&\; i = 1,2,..,M$$

Here $V_{I,i}$ and $V_{Q,i}$ are the amplitudes of the quadrature carriers amplitude modulated by the information symbols.

Baseband Rectangular M-QAM modulator:

There exist other constellation shapes that are more efficient (in terms of energy required to achieve same error probability) than the standard rectangular constellation. But due to its simplicity in modulation and demodulation rectangular constellations are preferred for implementation.

In practice, the information symbols are gray coded in-order to restrict the erroneous symbol decisions to single bit error, i.e, the adjacent symbols in the transmitter constellation should not differ more than one bit. Usually the gray coded symbols are separated into in-phase and quadrature bits and then mapped to M-QAM constellation. The rectangular configuration of QAM makes it easier to consolidate the previously mentioned steps into a simplified Look-Up-Table (LUT) approach.

16-QAM Modulation Scaling Factor:

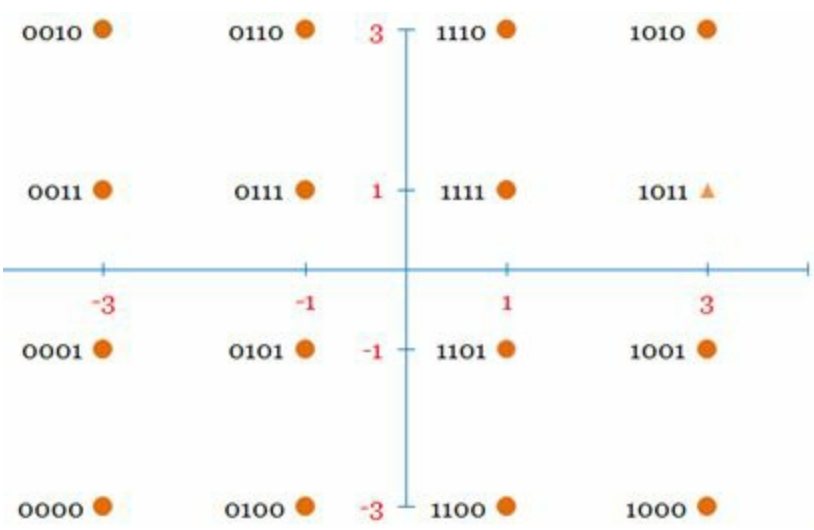
In order to get a fair comparison across all other modulations, the energy transmitted signal has to be normalized. In general the constellation points for a M-QAM modulation can be generated as

$$MQAM = a + jb \quad where\ a,b \in \{\pm 1, \pm 3, \dots, \pm(\lceil\sqrt{M}\rceil - 1)\}$$

The energy of a single constellation point is calculated as $E = V_{I,i}^2 + V_{Q,i}^2$ - where $V_{I,i}$ and $V_{Q,i}$ are I and Q components of the signaling point respectively. For a set of n constellation points, the total energy is calculated as

$$E = \sum_{i=1}^{n}(V_{I,i}^2 + V_{Q,i}^2)$$

In 16-QAM, there are 16 signal points in the constellation that are equally divided into four quadrants (each with four constellation points). Since the constellation is divided equally into four quadrants, normalizing the energy in a single quadrant will simplify things.



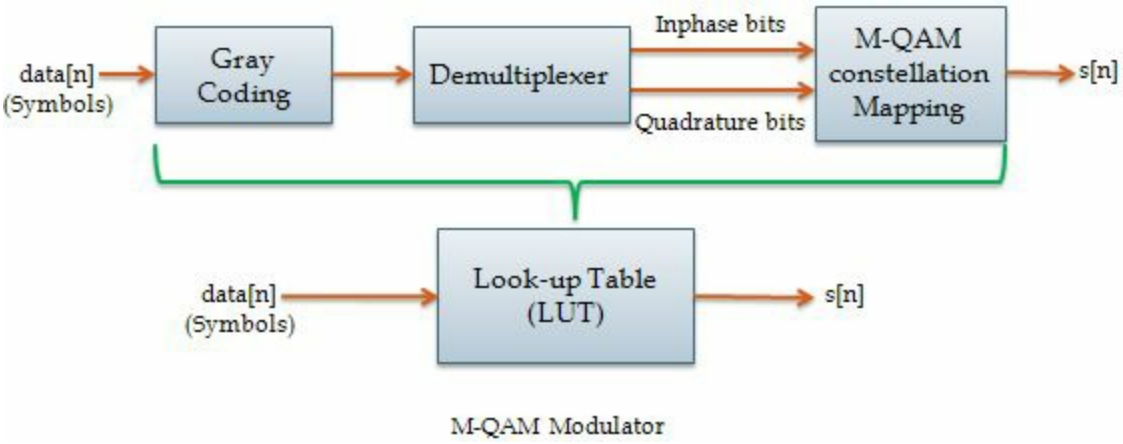
Calculating the total energy in any one of the quadrant, say for example -the top-right quadrant,

$$E = (1^2 + 1^2) + (1^2 + 3^2) + (3^2 + 1^2) + (3^2 + 3^2) = 40$$

The average energy is E_{avg}=E/4=40/4=10 and the normalization factor will be

$$Norm\ Factor = \frac{1}{\sqrt{E_{avg}}} = \frac{1}{\sqrt{10}}$$

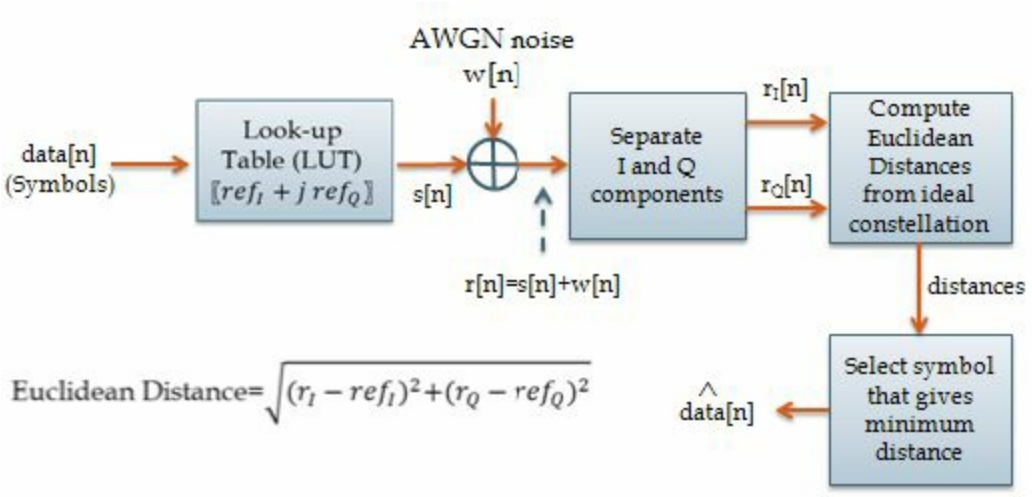
The values in the LUT (where the reference constellation is stored) are normalized by the above mentioned normalization factor and then the 16-QAM signal is generated.



M-QAM Modulator

Simulation Model:

The simulation model for M-QAM modulation is given in the next figure. The receiver uses Euclidean distance as a metric to decide on the received symbols.



Theoretical Symbol Error Rate:

The theoretical symbol error rate for M-QAM modulation technique in AWGN when $k=\log_2(M)$ is even, is given by

$$P_s \leq 1-\left[1-2\left(1-\frac{1}{\sqrt{M}}\right).Q\left(\sqrt{\frac{3kE_b}{(M-1)N_0}}\right)\right]^2$$

Or equivalently (Refer Appendix A6),

$$P_s \leq 1-\left[1-\left(1-\frac{1}{\sqrt{M}}\right).erfc\left(\sqrt{\frac{3kE_b}{2(M-1)N_0}}\right)\right]^2$$

Matlab Code:

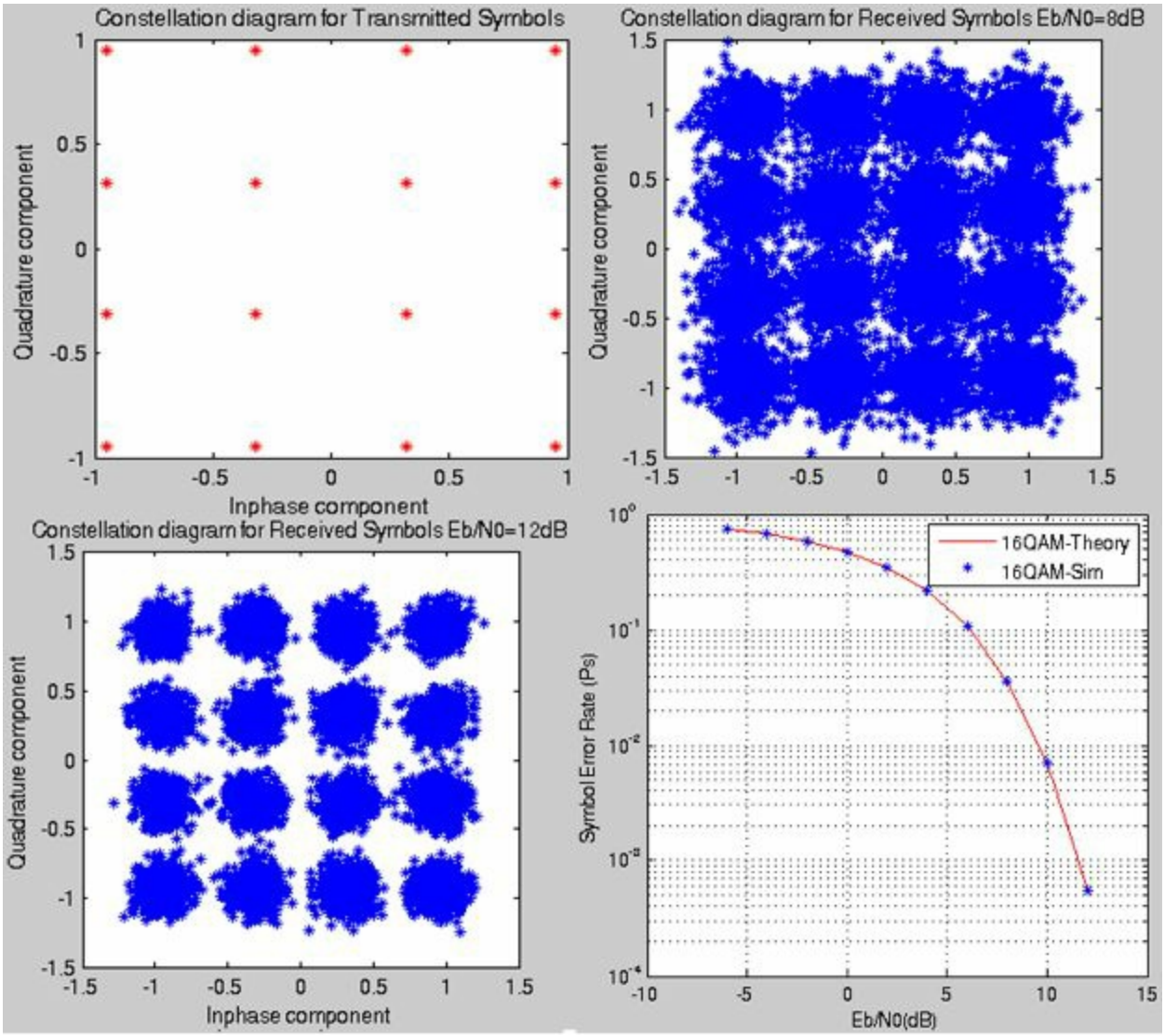
% Demonstration of Eb/N0 Vs SER for M-QAM modulation scheme

```
clear;clc;
%---------Input Fields-----------------------
N=10000; %Number of input symbols
EbN0dB = -6:2:12; %Define EbN0dB range for simulation
M=16; %for 16-QAM modulation.
%----------------------------------------

refArray =1/sqrt(10)*[-3-3j,-3-1j,-3+3j,-3+1j,-1-3j,-1-1j,-1+3j,-1+1j,3-3j,3-1j,3+3j,3+1j,1-3j,1-
1j,1+3j,1+1j];
symErrSimulated = zeros(1,length(EbN0dB));
k=log2(M);
EsN0dB = EbN0dB + 10*log10(k);

%---Generating a uniformly distributed random numbers in the set [0,1,2,..,M-1]
data=ceil(M.*rand(N,1))-1;
s=refArray(data+1); %16-QAM Constellation mapping with Gray coding

%--- Reference Constellation for demodulation and Error rate computation--
```

```
refI = real(refArray);
refQ = imag(refArray);

%---Place holder for Symbol Error values for each Es/N0 for particular M value--
index=1;
figure(1);
subplot(1,2,1);
plot(real(s),imag(s),'r*');
title('Constellation diagram for Transmitted Symbols');
xlabel('Inphase component');
ylabel('Quadrature component');
subplot(1,2,2);

for x=EsN0dB,
    %----------------------------------------
    %Channel Noise for various Es/N0
    %----------------------------------------
    %Adding noise with variance according to the required Es/N0
    noiseVariance = 1/(10.^(x/10));%Standard deviation for AWGN Noise
    noiseSigma = sqrt(noiseVariance/2);
    %Creating a complex noise for adding with M-QAM modulated signal
    %Noise is complex since M-QAM is in complex representation
    noise = noiseSigma*(randn(size(s))+1i*randn(size(s)));
    received = s + noise;

    %-------------I-Q Branching---------------
    r_i = real(received);
    r_q = imag(received);

    %---Decision Maker-Compute (r_i-s_i)^2+(r_q-s_q)^2 and choose the smallest
    r_i_repmat = repmat(r_i,M,1);
    r_q_repmat = repmat(r_q,M,1);
    plot(r_i,r_q,'*');
    title(['Constellation diagram for Received Symbols Eb/N0=' num2str(x-10*log10(k)) 'dB']);
    xlabel('Inphase component');
    ylabel('Quadrature component');
    pause;
    distance = zeros(M,N); %place holder for distance metric
    minDistIndex=zeros(N,1);

    for j=1:N
        %---Distance computation - (r_i-s_i)^2+(r_q-s_q)^2 --------------
        distance(:,j) = (r_i_repmat(:,j)-refI').^2+(r_q_repmat(:,j)-refQ').^2;
        %---capture the index in the array where the minimum distance occurs
        [dummy,minDistIndex(j)]=min(distance(:,j));
```

```
    end

    y = minDistIndex - 1;%The index becomes the decoded symbol

    %--------------Symbol Error Rate Calculation-----------------------------
    dataCap = y;
    symErrSimulated(1,index) = sum(dataCap~=data)/N;
    index=index+1;
end

%----- Compute Theoretical Symbol Error Rates --------------------
%EsN0lin = 10.^(EsN0dB/10);
EbN0lin = 10.^(EbN0dB/10);
symErrTheory = 2*(1-1/sqrt(M))*erfc(sqrt(3/2*k*EbN0lin/(M-1)));

%---------------Plotting commands----------------------
figure(2);
semilogy(EbN0dB,symErrTheory,'r-');hold on;
semilogy(EbN0dB,symErrSimulated,'b*');
legend('16QAM-Theory','16QAM-Sim');
xlabel('Eb/N0(dB)');
ylabel('Symbol Error Rate (Ps)');
grid on;
```

Simulation Results:



******

Back to Table of Contents

6.10 Symbol Error Rate Vs SNR performance curve simulation for 64-QAM

64-QAM Modulation Scaling Factor:

In order to get a fair comparison across all other modulations, the energy transmitted signal has to be normalized. In general the constellation points for a M-QAM modulation can be generated as

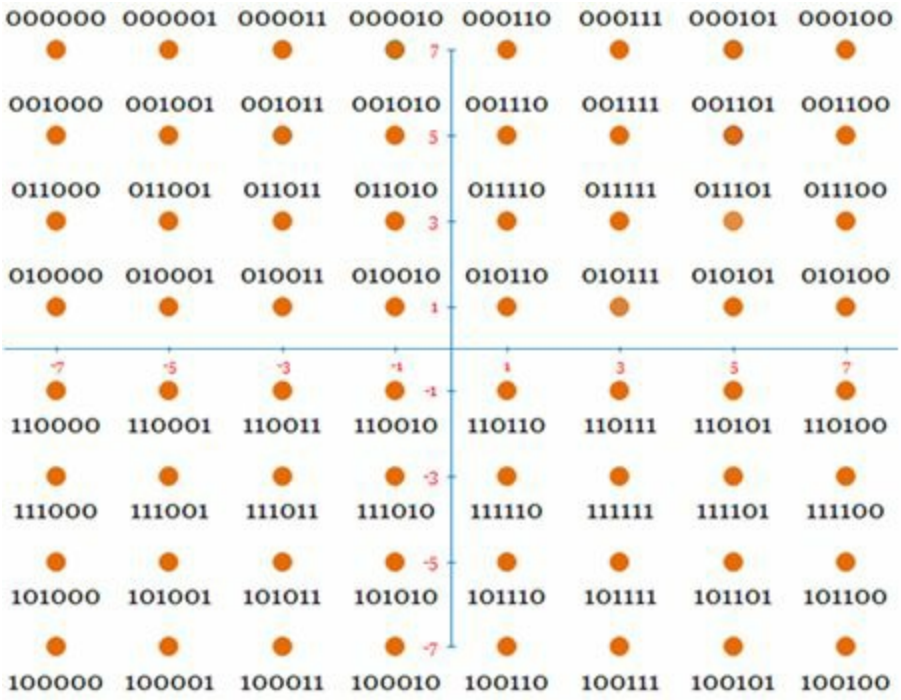
$$MQAM = a + jb \quad where\ a,b \in \{\pm 1, \pm 3, \dots, \pm(\lceil\sqrt{M}\rceil - 1)\}$$

The energy of a single constellation point is calculated as $E = V_{I,i}^2 + V_{Q,i}^2$ Where $V_{I,i}$ and $V_{Q,i}$ are the I and Q components of the signaling point. For a set of n constellation points, the total energy is calculated as

$$E = \sum_{i=1}^{n} (V_{I,i}^2 + V_{Q,i}^2)$$

In 64-QAM, there are 64 signal points in the constellation that are equally divided into four quadrants

(each with sixteen constellation points). Since the constellation is divided equally into four quadrants, normalizing the energy in a single quadrant will simplify things.



Calculating the total energy in any one of the quadrant, say for example -the top-right quadrant,

$$\begin{aligned} E = &(1^2+1^2)+(1^2+3^2)+(1^2+5^2)+(1^2+7^2) \\ &+(3^2+1^2)+(3^2+3^2)+(3^2+5^2)+(3^2+7^2) \\ &+(5^2+1^2)+(5^2+3^2)+(5^2+5^2)+(5^2+7^2) \\ &+(7^2+1^2)+(7^2+3^2)+(7^2+5^2)+(7^2+7^2) \\ &=672 \end{aligned}$$

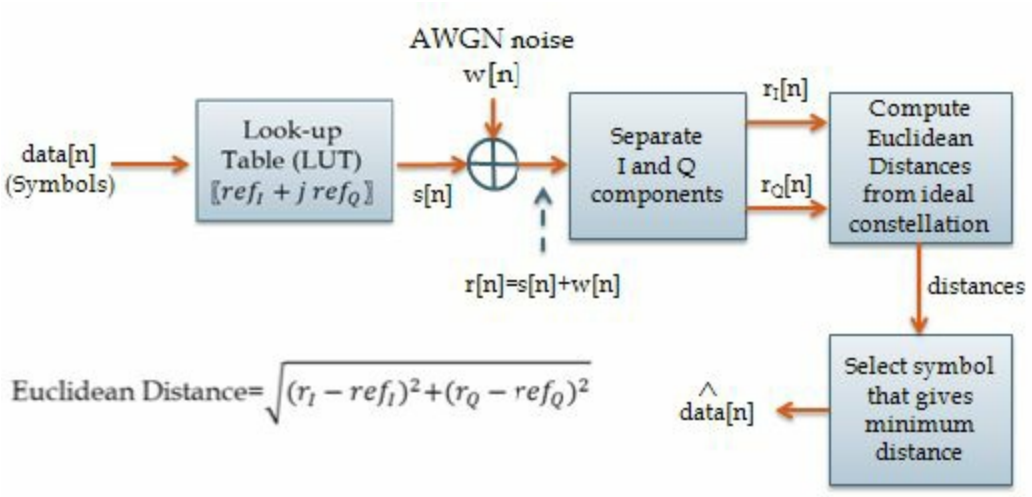
The average energy is E_{avg}=E/16=672/4=42 and the normalization factor will be

$$Norm\ Factor = \frac{1}{\sqrt{E_{avg}}} = \frac{1}{\sqrt{42}}$$

The values in the LUT (where the reference constellation is stored) are normalized by the above mentioned normalization factor and then the 64-QAM signal is generated.

Simulation Model:

The simulation model for M-QAM modulation is reproduced here for your convenience. The receiver uses Euclidean distance as a metric to decide on the received symbols.



Theoretical Symbol Error Rate:

The theoretical symbol error rate for M-QAM modulation technique in AWGN when k=log2(M) is even, is given by

$$P_s \leq 1-\left[1-2\left(1-\frac{1}{\sqrt{M}}\right).Q\left(\sqrt{\frac{3kE_b}{(M-1)N_0}}\right)\right]^2$$

Or equivalently (Refer Appendix A6),

$$P_s \leq 1-\left[1-\left(1-\frac{1}{\sqrt{M}}\right).erfc\left(\sqrt{\frac{3kE_b}{2(M-1)N_0}}\right)\right]^2$$

Matlab Code:

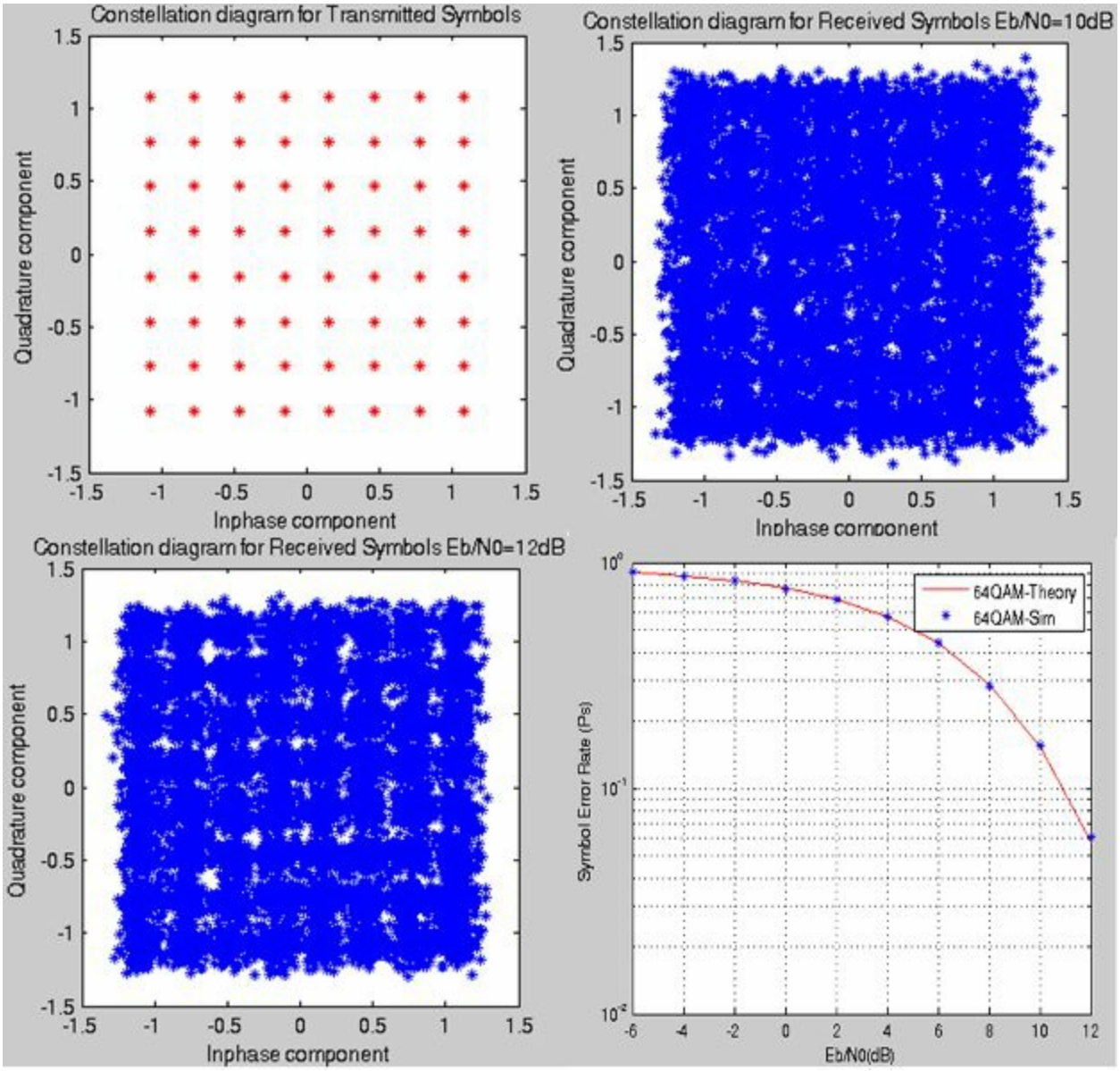
% Demonstration of Eb/N0 Vs SER for M-QAM modulation scheme

```
clear;clc;
%---------Input Fields-----------------------
N=10000; %Number of input symbols
EbN0dB = -6:2:12; %Define EbN0dB range for simulation

%------------------------------------------
M=64; %for 64-QAM modulation.
refArray =1/sqrt(42)*[-7+7j,-5+7j,-1+7j,-3+7j,7+7j,5+7j,1+7j,3+7j,-7+5j,-5+5j,-1+5j,-
3+5j,7+5j,5+5j,1+5j,3+5j,-7+j, -5+j, -1+j, -3+j, 7+j, 5+j, 1+j, 3+j,-7+3j,-5+3j,-1+3j,-
3+3j,7+3j,5+3j,1+3j,3+3j,-7-7j,-5-7j,-1-7j,-3-7j,7-7j,5-7j,1-7j,3-7j,-7-5j,-5-5j,-1-5j,-3-5j,7-5j,5-
5j,1-5j,3-5j,-7-j, -5-j, -1-j, -3-j, 7-j, 5-j, 1-j, 3-j,-7-3j,-5-3j,-1-3j,-3-3j,7-3j,5-3j,1-3j,3-3j];
symErrSimulated = zeros(1,length(EbN0dB));
k=log2(M);
EsN0dB = EbN0dB + 10*log10(k);

%---Generating a uniformly distributed random numbers in the set [0,1,2,..,M-1]
data=ceil(M.*rand(N,1))-1;
s=refArray(data+1); %64-QAM Constellation mapping with Gray coding
```

```
%--- Reference Constellation for demodulation and Error rate computation--
refI = real(refArray);
refQ = imag(refArray);

%---Place holder for Symbol Error values for each Es/N0 for particular M value--
index=1;
figure(1);
subplot(1,2,1);
plot(real(s),imag(s),'r*');
title('Constellation diagram for Transmitted Symbols');
xlabel('Inphase component');
ylabel('Quadrature component');
subplot(1,2,2);

for x=EsN0dB,
    %----------------------------------------
    %Channel Noise for various Es/N0
    %----------------------------------------
    %Adding noise with variance according to the required Es/N0
    noiseVariance = 1/(10.^(x/10));%Standard deviation for AWGN Noise
    noiseSigma = sqrt(noiseVariance/2);
    %Creating a complex noise for adding with M-QAM modulated signal
    %Noise is complex since M-QAM is in complex representation
    noise = noiseSigma*(randn(size(s))+1i*randn(size(s)));
    received = s + noise;

    %-------------I-Q Branching---------------
    r_i = real(received);
    r_q = imag(received);
    %---Decision Maker-Compute (r_i-s_i)^2+(r_q-s_q)^2 and choose the smallest
    r_i_repmat = repmat(r_i,M,1);
    r_q_repmat = repmat(r_q,M,1);
    plot(r_i,r_q,'*');
    title(['Constellation diagram for Received Symbols Eb/N0=' num2str(x-10*log10(k)) 'dB']);
    xlabel('Inphase component');
    ylabel('Quadrature component');
    pause;
    distance = zeros(M,N); %place holder for distance metric
    minDistIndex=zeros(N,1);

    for j=1:N
        %---Distance computation - (r_i-s_i)^2+(r_q-s_q)^2 --------------
        distance(:,j) = (r_i_repmat(:,j)-refI').^2+(r_q_repmat(:,j)-refQ').^2;
        %---capture the index in the array where the minimum distance occurs
```

```
        [dummy,minDistIndex(j)]=min(distance(:,j));
    end
    y = minDistIndex - 1;%The index becomes the decoded symbol

    %--------------Symbol Error Rate Calculation-----------------------------
    dataCap = y;
    symErrSimulated(1,index) = sum(dataCap~=data)/N;
    index=index+1;
end
%----- Compute Theoretical Symbol Error Rates --------------------
EbN0lin = 10.^(EbN0dB/10);
symErrTheory = (1-(1-(sqrt(M)-1)/sqrt(M)*erfc(sqrt(3/2*k*EbN0lin/(M-1))))).^2);

%---------------Plotting commands----------------------
figure(2);
semilogy(EbN0dB,symErrTheory,'r-');hold on;
semilogy(EbN0dB,symErrSimulated,'b*');
legend('64QAM-Theory','64QAM-Sim');
xlabel('Eb/N0(dB)');
ylabel('Symbol Error Rate (Ps)');
grid on;
```

Simulation Results:



******

Back to Table of Contents

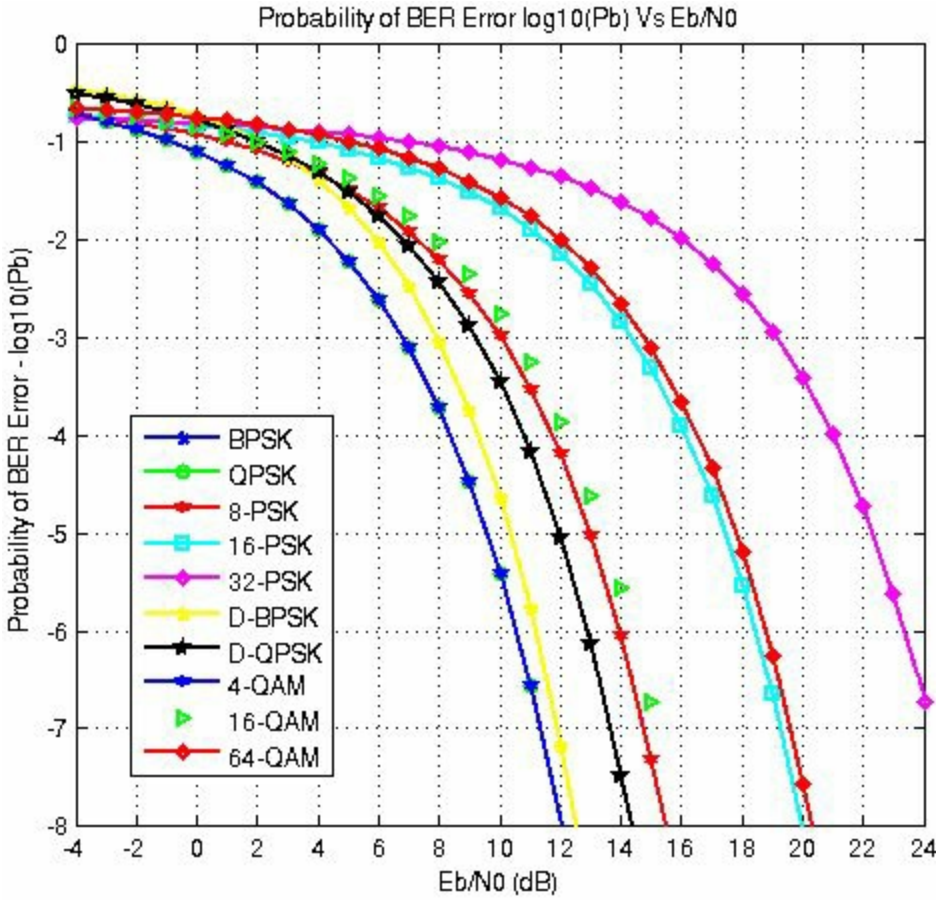
6.11 Performance comparison of Digital Modulation techniques

Let's take up some bandwidth-efficient linear digital modulation techniques (BPSK, QPSK and QAM) and compare its performance based on their theoretical BER over AWGN. (Readers are encouraged to read the previous chapter on Shannon's theorem and channel capacity (Section 1.5).

Following table summarizes the theoretical BER (given SNR per bit ration – E_b/N_0) for various linear modulations. Note that the E_b/N_0 values used in that table are in linear scale [to convert E_b/N_0 in dB to linear scale - use E_b/N_0 (linear) = 10^(E_b/N_0(dB)/10)].

Modulation	DetectionMethod	BitErrorRate (P_b)
BPSK	Coherent	$0.5erfc(\sqrt{\frac{E_b}{N_0}})$
QPSK	Coherent	$0.5erfc(\sqrt{\frac{E_b}{N_0}})$
$M-PSK$	Coherent	$\frac{1}{m}\,erfc\left(\sqrt{\frac{mE_b}{N_0}}sin(\frac{\pi}{M})\right)$
$M-QAM(m=even)$	Coherent	$\frac{2}{m}\left(1-\frac{1}{\sqrt{M}}\right)erfc(\sqrt{\frac{3mE_b}{2(M-1)N_0}})$
$D-BPSK$	$Non-coherent$	$0.5e^{-\frac{E_b}{N_0}}$
$D-QPSK$	$Non-coherent$	$Q_1(a,b)-0.5I_0(ab)e^{-0.5(a^2+b^2)}$ $where\ a=\sqrt{\frac{2E_b}{N_0}\left(1-\frac{1}{\sqrt{2}}\right)}$ $b=\sqrt{\frac{2E_b}{N_0}\left(1+\frac{1}{\sqrt{2}}\right)}$ $Q_1(a,b)$ = Marcum Q -function $I_0(ab)$ = Modified Bessel-function

A small script written in Matlab (given in the following text) gives the following output.



The following table is obtained by extracting the values of E_b/N_0 to achieve BER=10^{-6} from the previous figure. (Table data sorted with increasing values of E_b/N_0).

Modulation	Required Eb/N0 (dB)	Max $\eta=(\log_2(M))$ (bits/s/Hz)	C/N (dB)	Min Channel BW for ISI Free Signalling
BPSK	10.6	1	10.6	Rb
QPSK	10.6	2	13.6	0.5Rb
4-QAM	10.6	2	13.6	0.5Rb
D-BPSK	11.2	1	11.2	Rb
D-QPSK	12.7	2	15.7	0.5Rb
8-PSK	14	3	18.8	0.33Rb
16-QAM	14.5	4	20.5	0.25Rb
16-PSK	18.3	4	24.3	0.25Rb
64-QAM	18.8	6	26.6	0.17Rb
32-PSK	23.3	5	30.3	0.2Rb

In the table above η_B represents the spectral efficiency (otherwise called 'bandwidth efficiency') for the linear modulation with M constellation points and is given by

$$\eta_B = log_2(M) \quad bits/second/Hz$$

It represents the number of bits that can be stuffed in one symbol with data rate R_b (in bits/second) for a given minimum bandwidth.

$$B_{min} = \frac{R_b}{\eta_B}$$

If a pulse shaping technique like raised cosine pulse [with roll off factor (a)] is used then B_{min} becomes

$$B_{min} = (1 + a)\frac{R_b}{\eta_B}$$

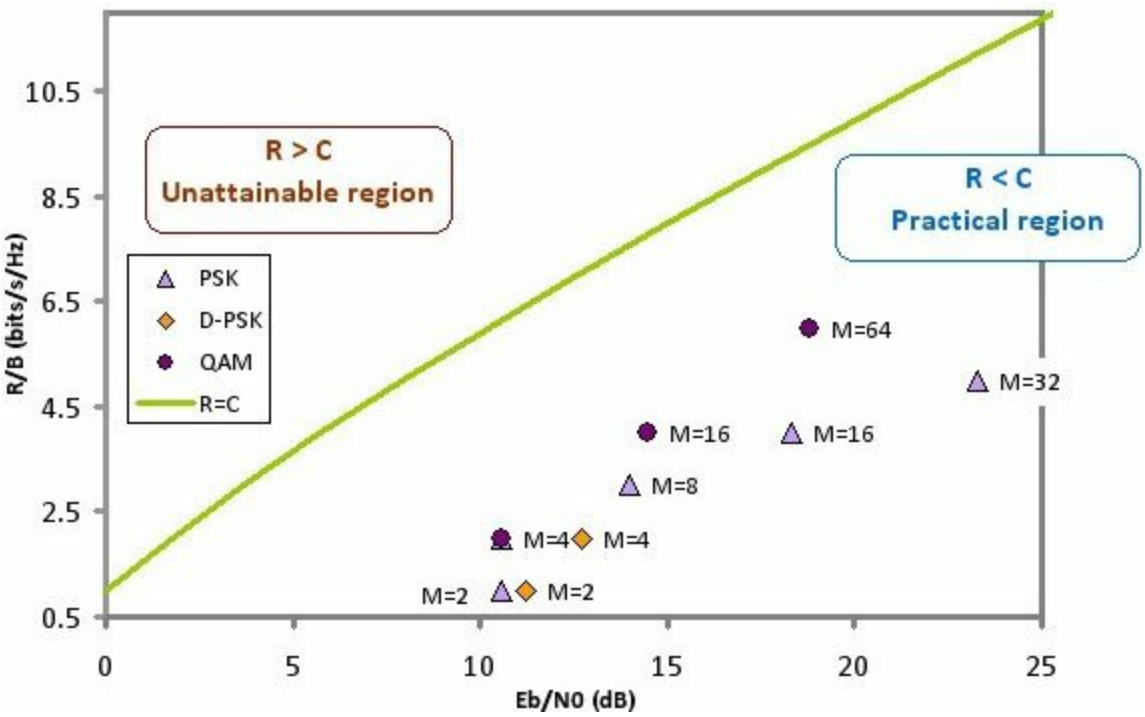
Next, the data in the previous table is plotted with E_b/N_0 on the x-axis and η on the y-axis (see the next figure) along with the well-known Shannon's Capacity equation over AWGN given by

$$C = B\ log\left(1 + \frac{C}{B}.\frac{E_b}{N_0}\right)$$

This can be represented as (refer the section on Unconstrained Shannon Limit for AWGN channel – Chapter 1.5])

$$\frac{E_b}{N_0} = \frac{2^{C/B} - 1}{C/B}$$

Plotting this relationship gives the following graph that shows the Shannon limit and the maximum achievable capacity of each modulation technique in an AWGN channel.



Matlab Code:

```
EbN0dB=-4:1:24;
EbN0lin=10.^(EbN0dB/10);
colors={'b-*','g-o','r-h','c-s','m-d','y-*','k-p','b-->','g:<','r-.d'};
index=1;

%BPSK
BPSK = 0.5*erfc(sqrt(EbN0lin));
plotHandle=plot(EbN0dB,log10(BPSK),char(colors(index)));
set(plotHandle,'LineWidth',1.5);
hold on;
index=index+1;

%M-PSK
m=2:1:5;
M=2.^m;
for i=M,
    k=log2(i);
    berErr = 1/k*erfc(sqrt(EbN0lin*k)*sin(pi/i));
    plotHandle=plot(EbN0dB,log10(berErr),char(colors(index)));
    set(plotHandle,'LineWidth',1.5);
    index=index+1;
end

%Binary DPSK
Pb = 0.5*exp(-EbN0lin);
plotHandle = plot(EbN0dB,log10(Pb),char(colors(index)));
set(plotHandle,'LineWidth',1.5);
```

```
index=index+1;

%Differential QPSK
a=sqrt(2*EbN0lin*(1-sqrt(1/2)));
b=sqrt(2*EbN0lin*(1+sqrt(1/2)));
Pb = marcumq(a,b,1)-1/2.*besseli(0,a.*b).*exp(-1/2*(a.^2+b.^2));
plotHandle = plot(EbN0dB,log10(Pb),char(colors(index)));
set(plotHandle,'LineWidth',1.5);
index=index+1;

%M-QAM
m=2:2:6;
M=2.^m;
for i=M,
    k=log2(i);
    berErr = 2/k*(1-1/sqrt(i))*erfc(sqrt(3*EbN0lin*k/(2*(i-1))));
    plotHandle=plot(EbN0dB,log10(berErr),char(colors(index)));
    set(plotHandle,'LineWidth',1.5);
    index=index+1;
end

legend('BPSK','QPSK','8-PSK','16-PSK','32-PSK','D-BPSK','D-QPSK','4-QAM','16-QAM','64-
QAM');
axis([-4 24 -8 0]);
set(gca,'XTick',-4:2:24); %re-name axis accordingly
ylabel('Probability of BER Error - log10(Pb)');
xlabel('Eb/N0 (dB)');
title('Probability of BER Error log10(Pb) Vs Eb/N0');
grid on;
```

******

Back to Table of Contents

6.12 Intuitive derivation of Performance of an optimum BPSK receiver in AWGN channel

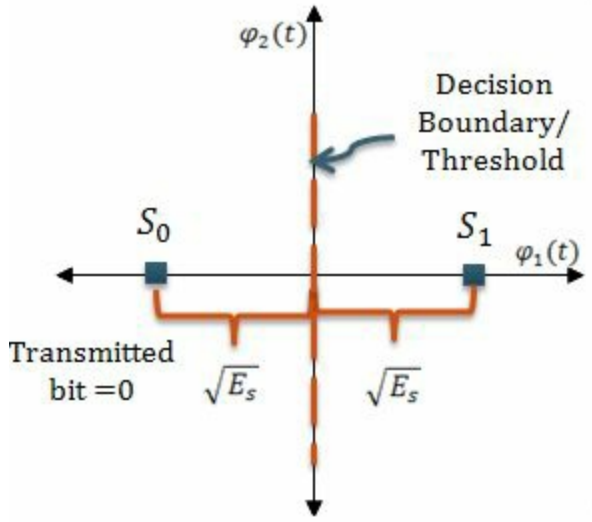
BPSK modulation is the simplest of all the M-PSK techniques. An insight into the derivation of error rate performance of an optimum BPSK receiver is essential as it serves as a stepping stone to understand the derivation for other comparatively complex techniques like QPSK,8-PSK etc..

Understanding the concept of Q function and error function (Refer Appendix A6) is a pre-requisite for this section.

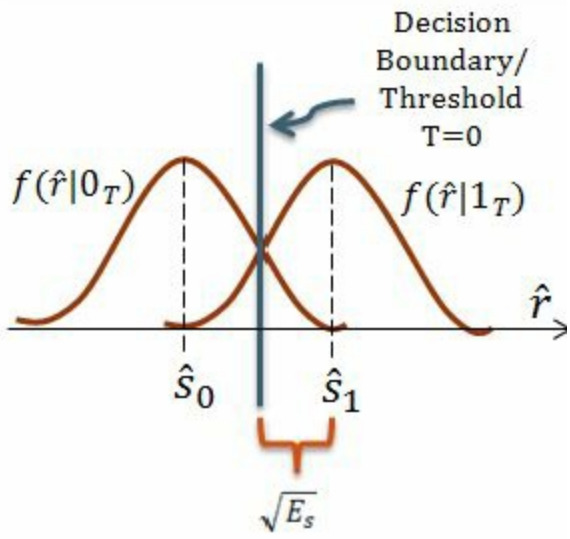
The ideal constellation diagram of a BPSK transmission (refer next figure) contains two constellation points located equidistant from the origin. Each constellation point is located at a distance $\sqrt{E_s}$ from

the origin, where E_S is the BPSK symbol energy. Since the number of bits in a BPSK symbol is always one, the notations – symbol energy (E_S) and bit energy (E_b) can be used interchangeably (E_S=E_b).

Assume that the BPSK symbols are transmitted through an AWGN channel characterized by variance = $N_0/2$ Watts. When 0 is transmitted, the received symbol is represented by a Gaussian random variable 'r' with mean=S_0 = $\sqrt{E_s}$ and variance =$N_0/2$. When 1 is transmitted, the received symbol is represented by a Gaussian random variable – r with mean=S_1= $\sqrt{E_s}$ and variance =$N_0/2$.



BPSK – ideal constellation



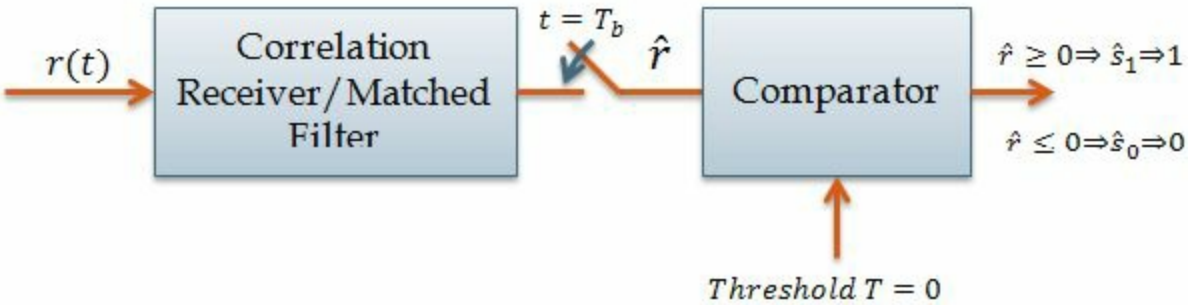
PDF of BPSK Symbols

Hence the conditional density function of the BPSK symbol (refer the figure above) is given by

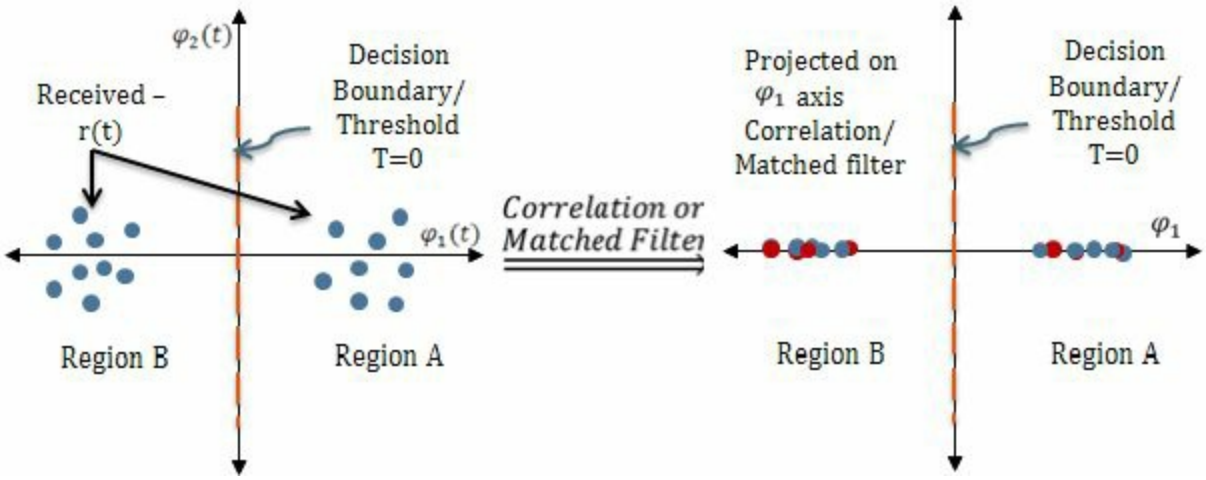
$$f(r \mid 0_T) = \frac{1}{\sqrt{\pi N_0}} exp\left\{-\frac{(\hat{r}-\hat{s}_0)^2}{N_0}\right\} \qquad (1A)$$

$$f(r \mid 1_T) = \frac{1}{\sqrt{\pi N_0}} exp\left\{-\frac{(\hat{r}-\hat{s}_1)^2}{N_0}\right\} \qquad (1B)$$

An optimum receiver for BPSK can be implemented using a correlation receiver or a matched filter receiver (Refer next Figure). Both these forms of implementations contain a decision making block that decides upon the bit/symbol that was transmitted based on the observed bits/symbols at its input.

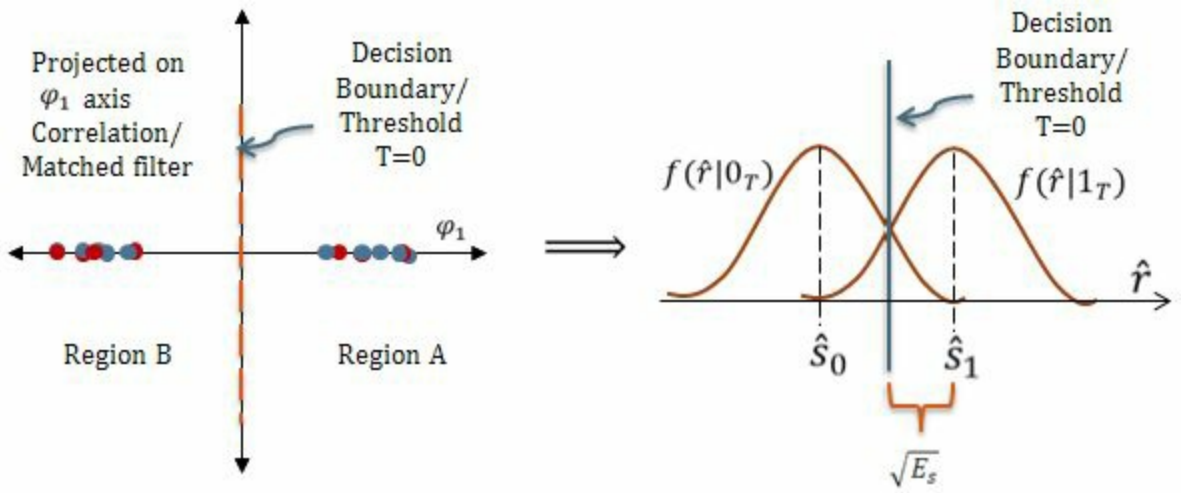


When the BPSK symbols are transmitted over an AWGN channel, the symbols appear smeared or distorted in the constellation depending on the SNR condition of the channel. A matched filter or that was previously used to construct the BPSK symbols at the transmitter. This process of projection is illustrated in the next figure.



Since the assumed channel is of Gaussian nature, the continuous probability density function of the projected bits will follow a Gaussian distribution. This is illustrated in the figure that follows.

After the signal points are projected on the basis function axis, a decision maker/comparator acts on those projected bits and decides on the fate of those bits based on the threshold set. For a BPSK receiver, if the a-prior probabilities of transmitted 0's and 1's are equal (P=0.5), then the decision boundary or threshold will pass through the origin. If the a-priori probabilities are not equal, then the optimum threshold boundary will shift away from the origin.



Considering a binary symmetric channel, where the a-priori probabilities of 0's and 1's are equal, the decision threshold can be conveniently set to T=0. The comparator decides whether the projected symbols are falling in region A or region B. If the symbols fall in region A, then it will decide that 1 was transmitted. It they fall in region B, the decision will be in favor of '0'.

For deriving the performance of the receiver, the decision process made by the comparator is applied to the underlying distribution model .The symbols projected on the axis will follow a Gaussian distribution. The threshold for decision is set to T=0. A received bit is in error, if the transmitted bit is '0' & the decision output is '1'. Similarly, the received bit is considered to be in error - if the transmitted bit is '1' & the decision output is '0'.

This is expressed in terms of probability of error as,

$$P(1\ decided, 0\ transmitted) + P(0\ decided, 1\ transmitted) \qquad (2)$$

Or equivalently,

$$P(error) = P(1_D, 0_T) + P(0_D, 1_T) \quad (3)$$

By applying Bayes Theorem, the above equation is expressed in terms of conditional probabilities as given below,

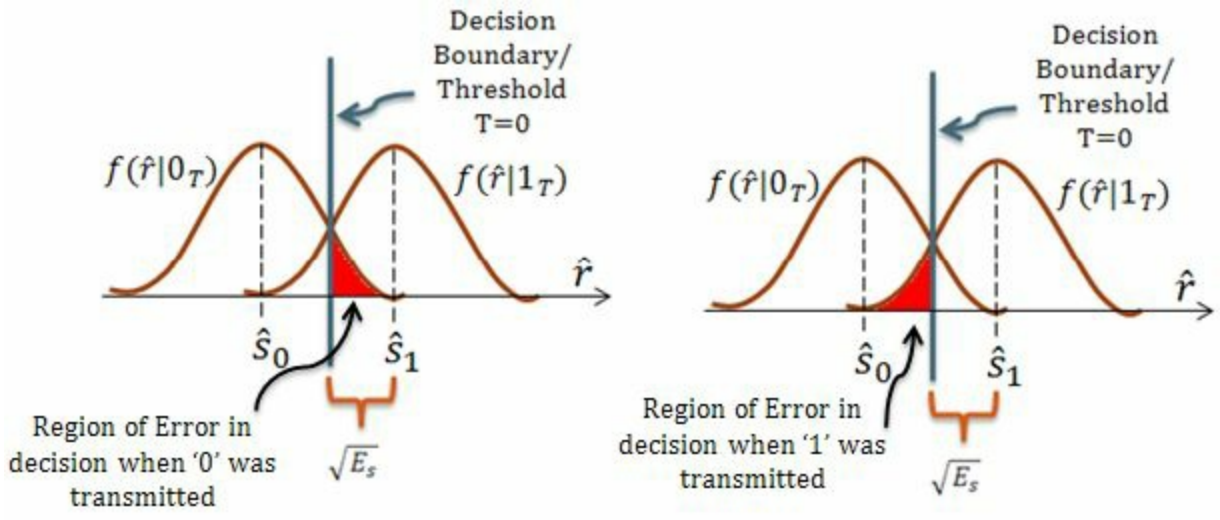
$$P(error) = P(0_T)P(1_D|0_T) + P(1_T)P(0_D|1_T) \quad (4)$$

$$P(error) = P(0_T)\int_0^\infty f(\hat{r}|0_T)d\hat{r} + P(1_T)\int_0^\infty f(\hat{r}|1_T)d\hat{r} \quad (5)$$

Since a-prior probabilities are equal $P(0_T)= P(1_T) =0.5$, the equation can be re-written as

$$P(error) = 0.5\int_0^\infty f(\hat{r}|0_T)d\hat{r} + 0.5\int_0^\infty f(\hat{r}|1_T)d\hat{r} \quad (6)$$

Intuitively, the integrals represent the area of shaded curves as shown in the next figure. From the discussion on "Q functions and error functions" (given in Appendix A6), we know that the area of the shaded region is given by Q function.



For the continuous density functions given in equation 1A, the mean=$S_0 =\sqrt{E_s}$ and variance $=N_0/2$, the area of region of error in the previous figure is given by

$$P(1_D|0_T) = Q\left(\sqrt{\frac{E_s}{N_0/2}}\right) \quad (7)$$

Similarly,

$$P(0_D|1_T) = Q\left(\sqrt{\frac{E_s}{N_0/2}}\right) \quad (8)$$

From (4), (6), (7) and (8),

$$P(error) = 0.5\,Q\left(\sqrt{\frac{E_s}{N_0/2}}\right) + 0.5\,Q\left(\sqrt{\frac{E_s}{N_0/2}}\right)$$
$$= Q\left(\sqrt{\frac{2E_s}{N_0}}\right) = 0.5\,erfc\left(\sqrt{\frac{E_s}{N_0}}\right) \qquad (9)$$

For BPSK, since $E_S=E_b$, the probability of symbol error (P_S) and the probability of bit error (P_b) are same. Therefore, expressing the P_S and P_b in terms of Q function and also in terms of complementary error function:

$$P_s = P_b = Q\left(\sqrt{\frac{2E_s}{N_0}}\right) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \qquad (10)$$

$$P_s = P_b = 0.5\,erfc\left(\sqrt{\frac{E_s}{N_0}}\right) = 0.5\,erfc\left(\sqrt{\frac{E_b}{N_0}}\right) \qquad (11)$$

******

Back to Table of Contents

Chapter 7: Orthogonal Frequency Division Multiplexing (OFDM)

7.1 Introduction to OFDM

In modulations, information is mapped on to changes in frequency, phase or amplitude (or a combination of them) of a carrier signal. Multiplexing deals with allocation/accommodation of users in a given bandwidth (i.e. it deals with allocation of available resource).

OFDM is a combination of modulation and multiplexing. In this technique, the given resource (bandwidth) is shared among individual modulated data sources. Normal modulation techniques (like AM, PM, FM, BPSK, QPSK, etc..,) are single carrier modulation techniques, in which the incoming information is modulated over a single carrier. OFDM is a multicarrier modulation technique, which employs several carriers, within the allocated bandwidth, to convey the information from source to destination. Each carrier may employ one of the several available digital modulation techniques (BPSK, QPSK, QAM etc..,).

Why OFDM

OFDM is very effective for communication over channels with frequency selective fading (different frequency components of the signal experience different fading). It is very difficult to handle frequency selective fading in the receiver, in which case, the design of the receiver is hugely complex. Instead of trying to mitigate frequency selective fading as a whole (which occurs when a huge bandwidth is allocated for the data transmission over a frequency selective fading channel), OFDM mitigates the problem by converting the entire frequency selective fading channel into small flat fading channels (as seen by the individual subcarriers). Flat fading is easier to combat (when

compared to frequency selective fading) by employing simple error correction and equalization schemes.

Difference between FDM and OFDM:

OFDM is a special case of FDM (Frequency Division Multiplexing). In FDM, the given bandwidth is subdivided among a set of carriers. There is no relationship between the carrier frequencies in FDM. For example, consider that the given bandwidth has to be divided among 5 carriers (say a, b, c, d, e). There is no relationship between the subcarriers; a, b, c, d and e can anything within the given bandwidth.

If the carriers are harmonics, say (b=2a, c=3a, d=4a, e=5a, integral multiples of fundamental component a) then they become orthogonal. This is a special case of FDM, which is called OFDM (as implied by the word – 'orthogonal' in OFDM)

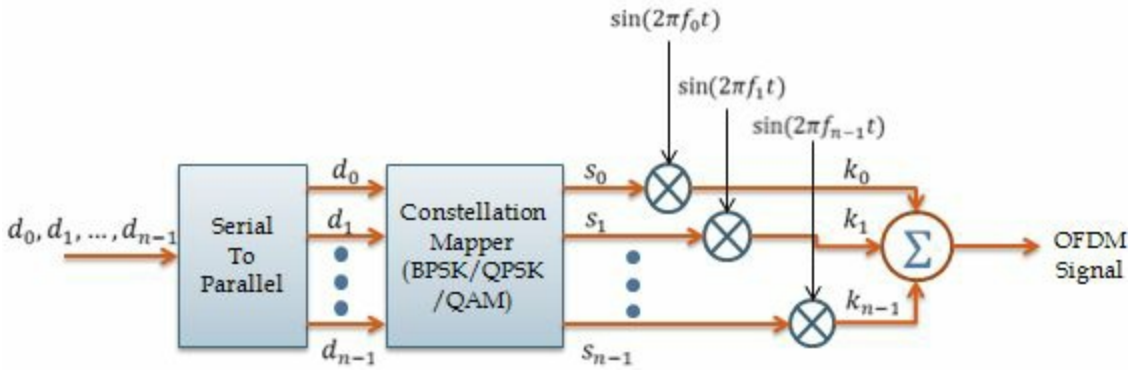
Designing OFDM Transmitter:

Consider that we want to send the following data bits using OFDM: D = $\{d_0,d_1,d_2,\ldots)$. The first thing that should be considered in designing the OFDM transmitter is the number of subcarriers required to send the given data. As a generic case, let's assume that we have N subcarriers. Each subcarrier is centered at frequencies that are orthogonal to each other (usually multiples of frequencies).

The second design parameter could be the modulation format that we wish to use. An OFDM signal can be constructed using anyone of the following digital modulation techniques namely BPSK, QPSK, QAM etc..,

The data (D) has to be first converted from serial stream to parallel stream depending on the number of sub-carriers (N). Since we assumed that there are N subcarriers allowed for the OFDM transmission, we name the subcarriers from 0 to N-1. Now, the Serial to Parallel converter takes the serial stream of input bits and outputs N parallel streams (indexed from 0 to N-1). These parallel streams are individually converted into the required digital modulation format (BPSK, QPSK, QAM etc..,). Let us call this output $S_0,S_1,..S_N$. The conversion of parallel data (D) into the digitally modulated data (S) is usually achieved by a constellation mapper, which is essentially a look up table (LUT). Once the data bits are converted to required modulation format, they need to be superimposed on the required orthogonal subcarriers for transmission. This is achieved by a series of N parallel sinusoidal oscillators tuned to N orthogonal frequencies ($f_0,f_1,\ldots f_{N-1}$). Finally, the resultant output from the N parallel arms are summed up together to produce the OFDM signal.

The following figure illustrates the basic concept of OFDM transmission (Note: In order to give a simple explanation to illustrate the underlying concept, the usual IFFT/FFT blocks that are used in actual OFDM system, are not used in the block diagram) .



Example:

The first example illustrates the concept of OFDM transmission with BPSK modulation as its underlying modulation format. The second example illustrates the OFDM transmission with pi/4 shifted QPSK modulation. Here five orthogonal subcarriers are assumed for the OFDM transmission.

Data = 1,0,1,0,1,1,1,1,0,1,0,0,0,1,0,1,0,0,1,1,

Time	D_0	d_1	d_2	d_3	d_4
t_0	1	0	1	0	1
t_1	1	1	1	0	1
t_2	0	0	0	1	0
t_3	1	0	0	1	1

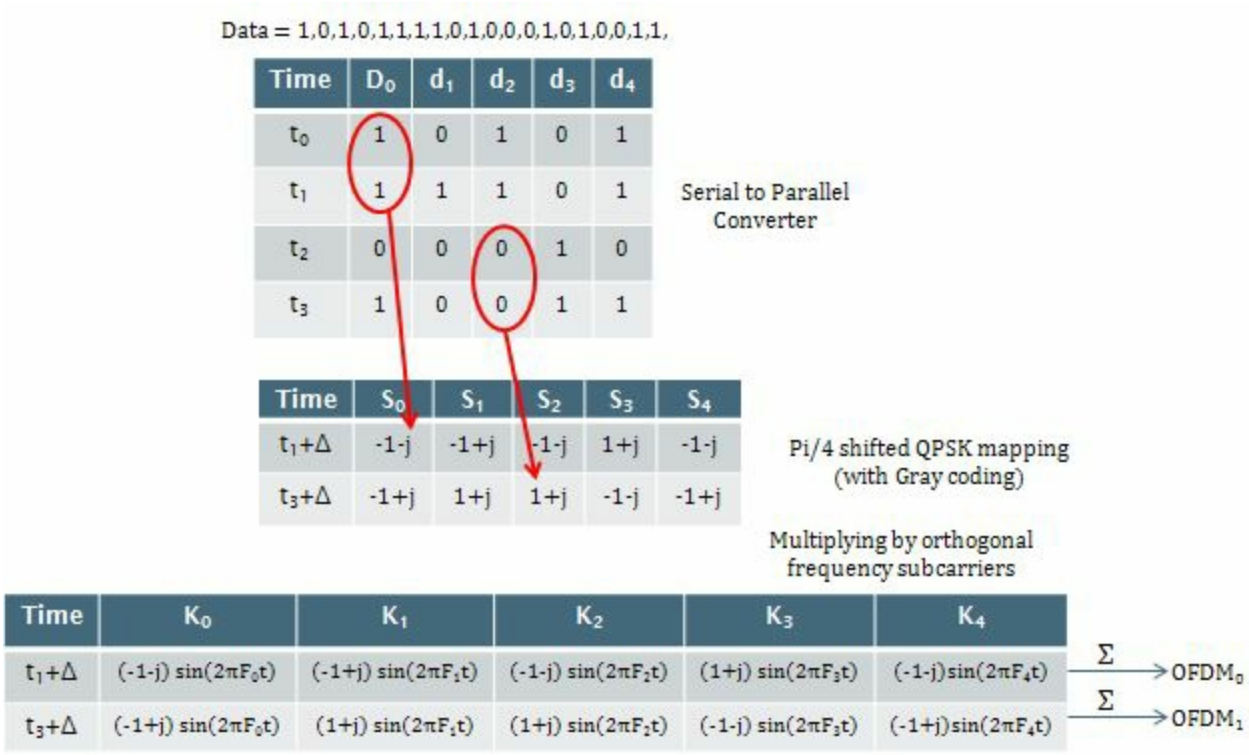
Serial to Parallel Converter

Time	S_0	S_1	S_2	S_3	S_4
t_0	1	-1	1	-1	1
t_1	1	1	1	-1	1
t_2	-1	-1	-1	1	-1
t_3	1	-1	-1	1	1

BPSK Mapping

Multiplying by orthogonal frequency subcarriers

Time	K_0	K_1	K_2	K_3	K_4	
$t_0+\Delta$	$1\times\sin(2\pi F_0 t)$	$-1\times\sin(2\pi F_1 t)$	$1\times\sin(2\pi F_2 t)$	$-1\times\sin(2\pi F_3 t)$	$1\times\sin(2\pi F_4 t)$	$\Sigma \rightarrow OFDM_0$
$t_1+\Delta$	$1\times\sin(2\pi F_0 t)$	$1\times\sin(2\pi F_1 t)$	$1\times\sin(2\pi F_2 t)$	$-1\times\sin(2\pi F_3 t)$	$1\times\sin(2\pi F_4 t)$	$\Sigma \rightarrow OFDM_1$
$t_2+\Delta$	$-1\times\sin(2\pi F_0 t)$	$-1\times\sin(2\pi F_1 t)$	$-1\times\sin(2\pi F_2 t)$	$1\times\sin(2\pi F_3 t)$	$-1\times\sin(2\pi F_4 t)$	$\Sigma \rightarrow OFDM_2$
$t_3+\Delta$	$1\times\sin(2\pi F_0 t)$	$-1\times\sin(2\pi F_1 t)$	$-1\times\sin(2\pi F_2 t)$	$1\times\sin(2\pi F_3 t)$	$1\times\sin(2\pi F_4 t)$	$\Sigma \rightarrow OFDM_3$



******

Back to Table of Contents

7.2 Role of FFT/IFFT in OFDM

In the previous section, the architecture of an OFDM transmitter was described using sinusoidal components. Generally, an OFDM signal can be represented as

$$OFDM\ signal = c(t) = \sum_{n=0}^{N-1} s_n(t)\sin(2\pi f_n t)$$

Here, s(t) represents the symbols mapped to chosen constellation (BPSK/QPSK/QAM etc..,) and F_n represents the orthogonal frequencies.

This equation can be thought of as an IFFT process (Inverse Fast Fourier Transform). The Fourier transform breaks a signal into different frequency bins by multiplying the signal with a series of sinusoids. This essentially translates the signal from time domain to frequency domain. But, we always view IFFT as a conversion process from frequency domain to time domain.

FFT is represented by

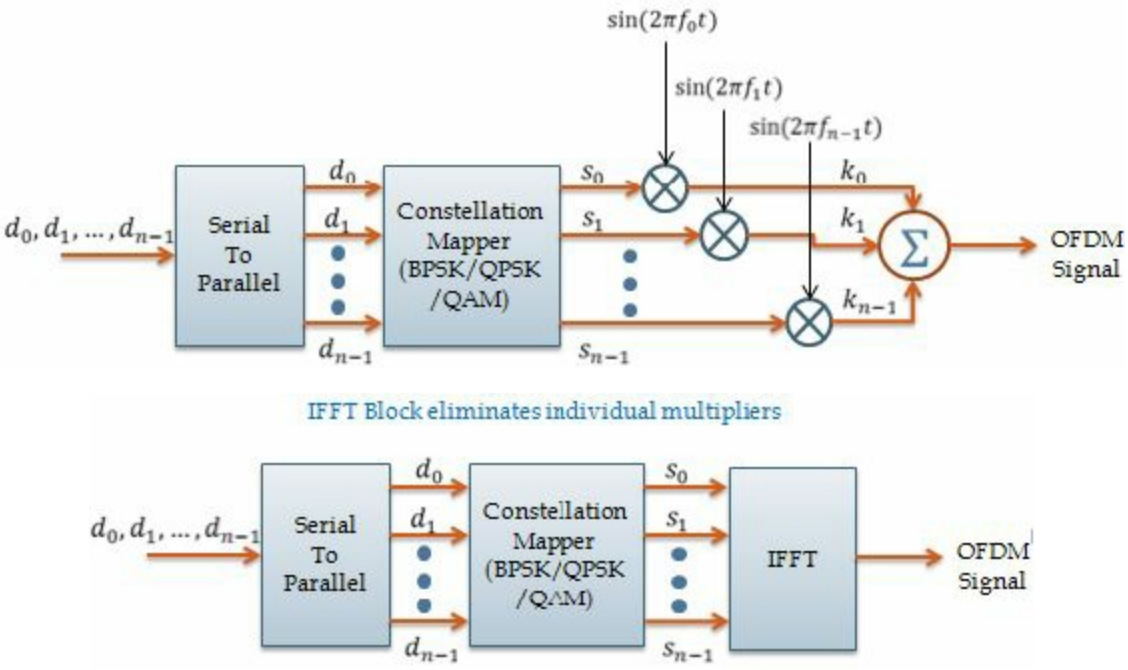
$$X(k) = \sum_{n=0}^{N-1} x(n)sin\left(\frac{2\pi kn}{N}\right) + j\sum_{n=0}^{N-1} x(n)cos\left(\frac{2\pi kn}{N}\right)$$

whereas its dual, IFFT is given by

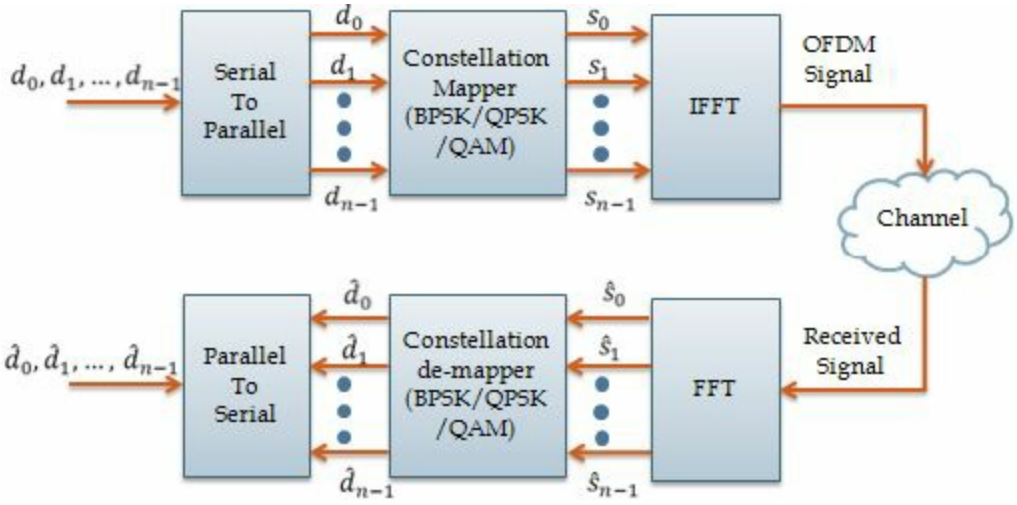
$$x(n) = \sum_{n=0}^{N-1} X(k) sin\left(\frac{2\pi kn}{N}\right) - j \sum_{n=0}^{N-1} X(k) cos\left(\frac{2\pi kn}{N}\right)$$

The equation for FFT and IFFT differ by the coefficients they take and the minus sign. Both equations do the same thing. They multiply the incoming signal with a series of sinusoids and separate them into bins. In fact, FFT and IFFT are dual and behave in a similar way. IFFT and FFT blocks are interchangeable.

Since the OFDM signal (c(t) in the equation above) is in time domain, IFFT is the appropriate choice to use in the transmitter, which can be thought of as converting frequency domain samples to time domain samples. Well, you might ask: s(t) is not in frequency domain and they are already in time domain; so, what is the need to convert it into time domain again ? The answer is IFFT/FFT equation comes handy in implementing the conversion process and we can eliminate the individual sinusoidal multipliers required in the transmitter/receiver side. The following figure illustrates how the use of IFFT in the transmitter eliminates the need for separate sinusoidal converters. Always remember that IFFT and FFT blocks in the transmitter are interchangeable as long as their duals are used in receiver.



The entire architecture of a basic OFDM system with both transmitter and receiver will look like the one given in the following figure



An OFDM system is defined by IFFT/FFT length – N, the underlying modulation technique (BPSK/QPSK/QAM), supported data rate, etc. The FFT/IFFT length N defines the number of total subcarriers present in the OFDM system. For example, an OFDM system with N=64 provides 64 subcarriers. In reality, not all the subcarriers are utilized for data transmission. Some subcarriers are reserved for pilot carriers (used for channel estimation/equalization and to combat magnitude and phase errors in the receiver) and some are left unused to act as guard band. OFDM systems do not transmit any data on the subcarriers that are near the two ends of the transmission band (Not necessarily at the ends of the bands, implementations may differ). These subcarriers are collectively called guard band. The reservation of subcarriers to guard bands helps to reduce the out of band radiation and thus eases the requirements on transmitter front-end filters. The subcarriers in the guard band are also called Null subcarriers or virtual subcarriers.

For example IEEE 802.11 standard [IEEE80211] specifies the following parameters for its OFDM physical layer.

FFT/IFFT size = 64 (implies 64 subcarriers in total = used + unused = N_{FFT})
Number of data subcarriers = 48 (Nd)
Number of pilot subcarriers = 4 (Np)
Derived parameters from the above specification are:
Number of total USED subcarriers = 52 (Nu = Nd+ Np)
Number of total UNUSED subcarriers = 12 (Nun = N_{FFT} – Nu).

According to the spec, these 52 used subcarriers are distributed in the following way. The 52 used subcarriers are named as 1, 2, 3… 26 and -1, -2, -3… -26. The used subcarriers 1 to 26 are mapped to 1 to 26 of IFFT inputs and the subcarriers -1,-2, ..., -26 are mapped to the IFFT inputs 38 to 63. The remaining IFFT inputs 27 to 37 and the input 0 (dc input) are set to “0”. In this manner the 12 null subcarriers are mapped to IFFT inputs.

******

Back to Table of Contents

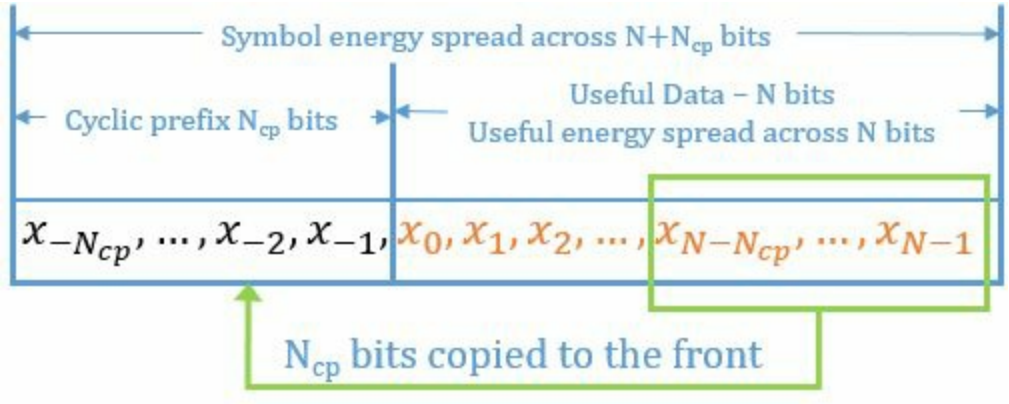
7.3 Role of Cyclic Prefix in OFDM

Consider a non-ideal channel h(t)≠δ (t), that causes delay dispersion. Delay dispersion manifests itself as Inter Symbol Interference (ISI) on each subcarrier channel due to pulse overlapping. It will also cause Inter Carrier Interference (ICC) due to the non-orthogonality of the received signal. Adding cyclic prefix to each OFDM symbol mitigates the problems of ISI and ICC by removing them altogether.

Let us say, without cyclic prefix we transmit the following N values (N=N_{FFT}=length of FFT/IFFT) for a single OFDM symbol.

$$X_0, X_1, X_2, \ldots, X_{N-1}$$

Let us consider a cyclic prefix of length Ncp, (where Ncp<N), is formed by copying the last Ncp values from the above vector of X and adding those Ncp values to the front part of the same X vector.



If T is the duration of the an OFDM symbol in seconds, due to the addition of cyclic prefix of length Ncp, the total duration of an OFDM symbol becomes T+Tcp, where Tcp=Ncp*T/N. Therefore, the number of samples allocated for cyclic prefix can be calculated as Ncp=Tcp*N/T, where N is the FFT/IFFT length, T is the IFFT/FFT period and Tcp is the duration of cyclic prefix.

The key ideas behind adding cyclic prefix:

1) Convert linear convolution into circular convolution which eases the process of detecting the received signal by using a simple single tap equalizer
If you wish to know how the addition of cyclic prefix converts linear convolution to circular convolution visit external link [TechnicalJar]

2) Help combat ISI and ICC.

When a cyclic prefix of length Ncp is added to the OFDM symbol, the output of the channel (r) is given by circular convolution of channel impulse response (h) and the OFDM symbols with cyclic prefix (x).

$$r = h \circledast x$$

As we know, for the discrete signals, circular convolution in the time domain translates to multiplication in the frequency domain. Thus, in frequency domain, the above equation translates to

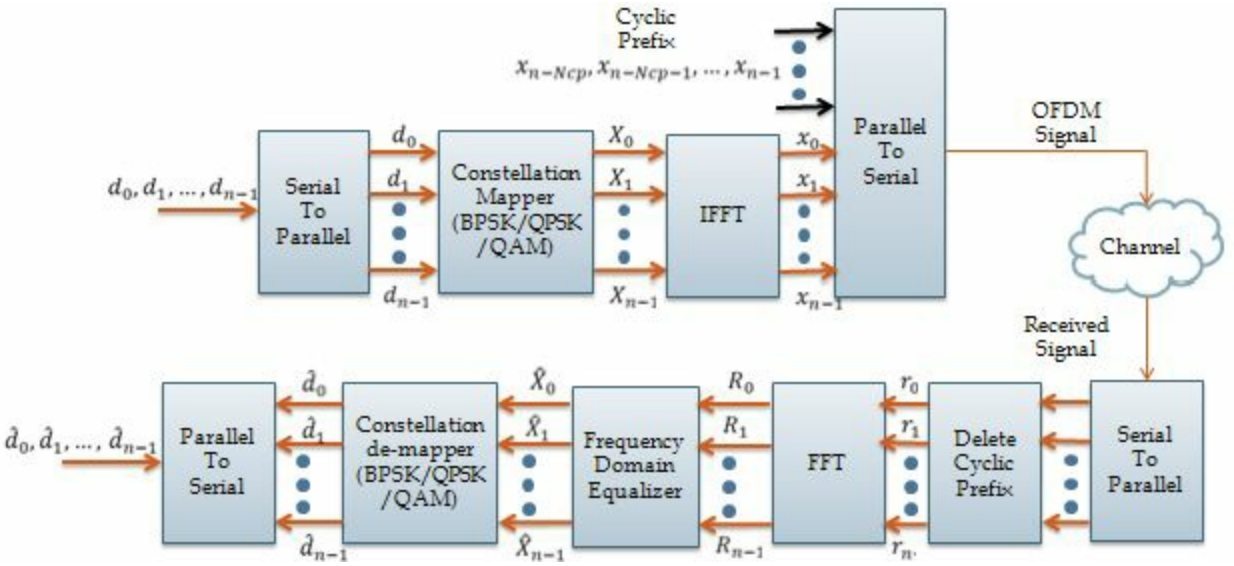
$$R = HX$$

At the receiver, R is the received signal (in Frequency domain) and our goal is to estimate the transmitted signal (X) from the received signal R. From the above equation, the problem of detecting the transmitted signal at the receiver side translates to a simple equalization equation as follows

$$\hat{X} = \frac{R}{H}$$

After the FFT is performed at the receiver side (i.e. after the FFT block in the receiver side), a single tap equalizer (which essentially implements the above equation) is used to estimate the transmitted OFDM symbol. It also corrects the phase and equalizes the amplitude.

A basic OFDM architecture with Cyclic Prefix is given below. (In the following diagram, symbols represented by small case letters are assumed to be in time domain, whereas the symbols represented by uppercase letters are assumed to be in frequency domain).



The IEEE specifications specify the length of the cyclic prefix in terms of its duration. Let us see how to convert the specified duration (Tcp) in to actual number of samples assigned for cyclic prefix (Ncp).

Let us see an example of deriving Ncp from IEEE 802.11 specifications [IEEE80211].

Given parameters in the spec:

N=64;%FFT size or total number of subcarriers (used + unused) 64
Nsd = 48;%Number of data subcarriers 48
Nsp = 4 ;%Number of pilot subcarriers 4
ofdmBW = 20 * 10^6 ; % OFDM bandwidth

Derived Parameters:

deltaF = ofdmBW/N;% Bandwidth for each subcarrier - include all used and unused subcarriers
Tfft = 1/deltaF;% IFFT or FFT period = 3.2us
Tgi = Tfft/4;% Guard interval duration - duration of cyclic prefix - 1/4th portion of OFDM symbols
Tsignal = Tgi+Tfft;% Total duration of BPSK-OFDM symbol = Guard time + FFT period
Ncp = N*Tgi/Tfft;%Number of symbols allocated to cyclic prefix
Nst = Nsd + Nsp;% Number of total used subcarriers
nBitsPerSym=Nst;%For BPSK the number of Bits per Symbol is same as number of subcarriers

******

Back to Table of Contents

7.4 Simulation of OFDM system in Matlab – BER Vs Eb/N0 for OFDM in AWGN channel

To simulate an OFDM system, following design parameters are essential. Let us consider the OFDM system parameters as defined in IEEE 802.11[IEEE80211] specifications. The sample parameters derived in the previous section are used here as inputs for simulation.

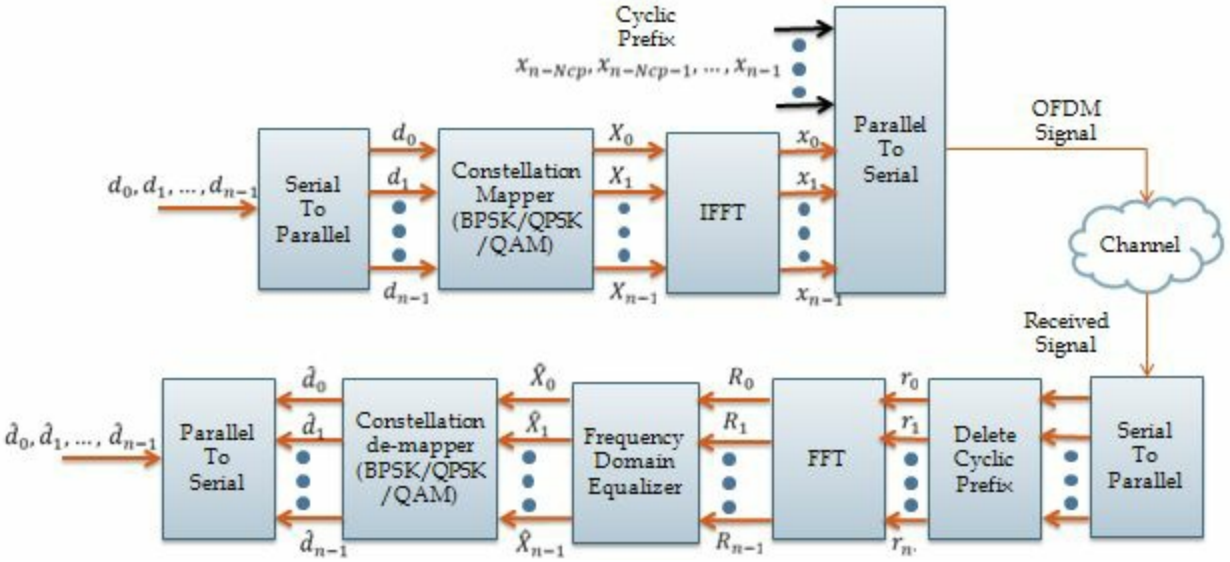
Given parameters in the spec:

N=64;%FFT size or total number of subcarriers (used + unused) 64
Nsd = 48;%Number of data subcarriers 48
Nsp = 4 ;%Number of pilot subcarriers 4
ofdmBW = 20 * 10^6 ; % OFDM bandwidth

Derived Parameters:

deltaF = ofdmBW/N;% Bandwidth for each subcarrier - include all used and unused subcarriers
Tfft = 1/deltaF;% IFFT or FFT period = 3.2us
Tgi = Tfft/4;% Guard interval duration - duration of cyclic prefix - 1/4th portion of OFDM symbols
Tsignal = Tgi+Tfft;% Total duration of BPSK-OFDM symbol = Guard time + FFT period
Ncp = N*Tgi/Tfft;%Number of symbols allocated to cyclic prefix
Nst = Nsd + Nsp;% Number of total used subcarriers
nBitsPerSym=Nst;%For BPSK the number of Bits per Symbol is same as number of subcarriers

We will use the following model for simulating the OFDM system.



The channel in this case is modeled as a simple AWGN channel. Since the channel is considered to be an AWGN channel, there is no need for the frequency domain equalizer in the OFDM receiver (Frequency domain equalizer will be helpful only if the channel introduces multipath fading). Since our channel is an AWGN channel, the frequency domain equalizer block in the above diagram can be removed.

Calculating E_S/N_0 or E_b/N_0 for OFDM system:

In order to do a Monte Carlo simulation of an OFDM system, required amount of channel noise has to be generated that is representative of the required E_b/N_0. In Matlab it is easier to generate a Gaussian noise with zero mean and unit variance. The generated zero-mean-unit-variance noise has to be scaled accordingly to represent the required E_b/N_0 or E_S/N_0. If we have E_S/N_0, the required noise can be generated from zero-mean-unit-variance-noise by,

$$required\ noise = 10^{-\frac{E_S}{N_0}\frac{1}{20}} \times noise$$

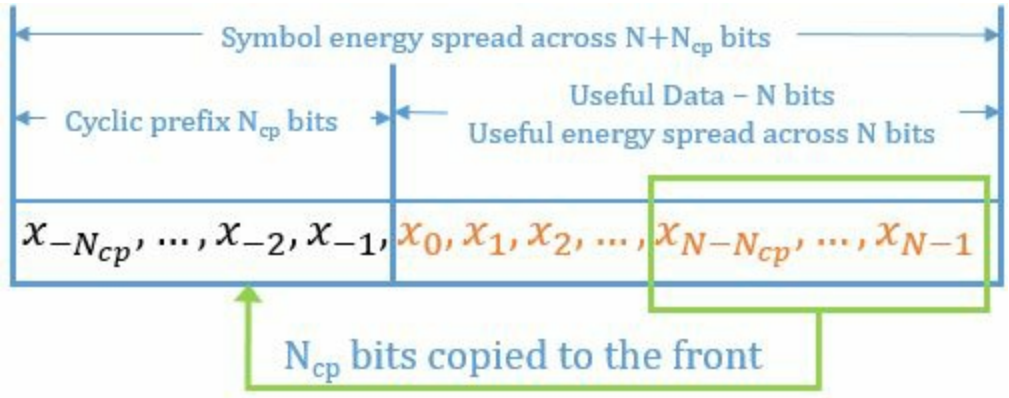
Since the OFDM system transmits and receives the data in symbols, it is appropriate/easier to

generate the required noise based on E_S/N_0 instead of E_b/N_0 (as given above). But we are interested in plotting BER against E_b/N_0. Ok!!! Then… how do we convert given E_b/N_0 to E_S/N_0 for an OFDM system?

Normally for a simple BPSK system, bit energy and symbol energy are the same. So E_b/N_0 and E_S/N_0 are the same for a BPSK system. But for an OFDM BPSK system, they are not the same. This is because; each OFDM symbol contains additional overhead in both time domain and frequency domain. In the time domain, the cyclic prefix is an additional overhead added to each OFDM symbol that is being transmitted. In the frequency domain, not all the subcarriers are utilized for transmitting the actual data bits, rather a few subcarriers are unused and are reserved as guard bands.

Effect of Cyclic Prefix on E_S/N_0:

The following diagram illustrates the concept of cyclic prefix. Each OFDM symbol contains both useful data and overhead (in the form of cyclic prefix). The bit energy represents the energy contained in the useful bits. In this case, the bit energy is spread over N bits (where N is the FFT size). On top of the useful data, additional Ncp bits are added as cyclic prefix, which forms the overhead. So if the entire OFDM symbol is considered, the symbol energy is spread across N+Ncp bits.



This relationship is given as

$$E_s(N + N_{cp}) = NE_b$$

which translates to,

$$E_s - \frac{N}{N + N_{cp}} E_b \qquad (1)$$

Effect of unused subcarriers on Es/N0:

As mentioned earlier, not all the subcarriers are used for transmission. Out of the total N subcarriers, only Nst carriers are used for the OFDM symbols transmission (this includes both data and pilot subcarriers). Again, in the frequency domain, the useful bit energy is spread across Nst subcarriers, whereas the symbol energy is spread across N subcarriers. This gives us another relationship between E_S and E_b as given below

$$E_s \times N = N_{st} \times E_b$$

which translates to,

$$E_s = \frac{N_{st}}{N} \times E_b \qquad (2)$$

From (1) and (2), the overall effect of both cyclic prefix and unused subcarriers on Es/N0 is given by

$$\frac{E_s}{N_0} = \left(\frac{N}{N_{cp}+N}\right)\left(\frac{N_{st}}{N}\right)\frac{E_b}{N_0} \qquad (3)$$

which, when converted to dB yields the following relationship.

$$\left(\frac{E_s}{N_0}\right)_{dB} = \left(\frac{N}{N_{cp}+N}\right)_{dB} + \left(\frac{N_{st}}{N}\right)_{dB} + \left(\frac{E_b}{N_0}\right)_{dB} \qquad (4)$$

Since Ncp- cyclic prefixes are added to the OFDM symbol, the output signal from the parallel to serial converter has to be boosted to compensate the wastage of energy associated with the addition of cyclic prefix. To properly generate the required SNR in Matlab, the signal term at the output of the parallel to serial converter has to be scaled as follows

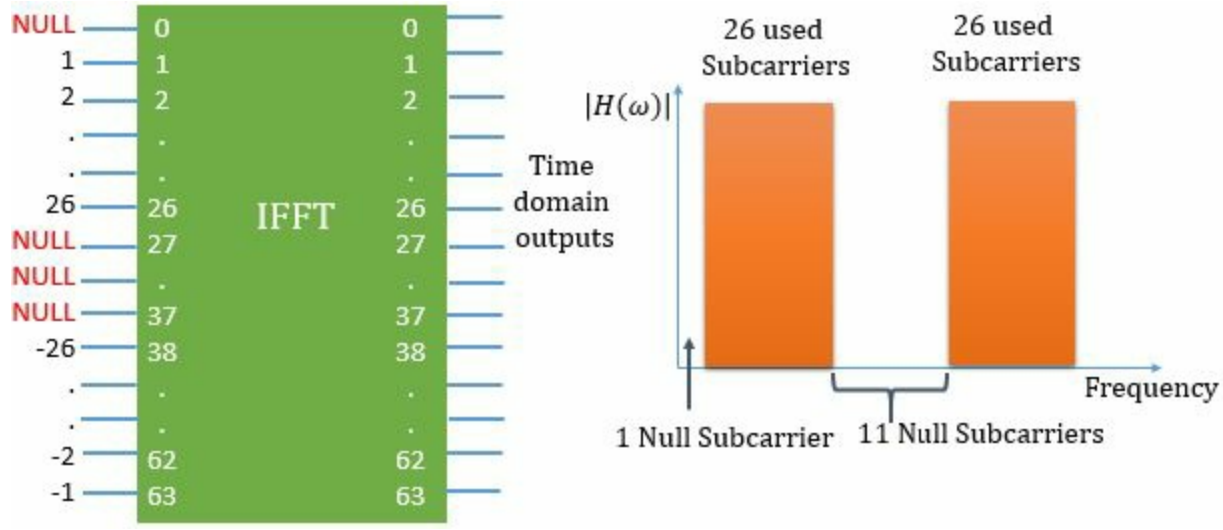
$$Boosted\ OFDM\ Signal = \sqrt{\frac{N_{cp}+N}{N}} \times OFDM\ signal$$

The received signal is represented as (for the given E_b/N_0)

$$y = boosted\ OFDM\ signal + required\ noise$$

Arrangement of subcarriers:

The IEEE 802.11 specification specifies how to arrange the given subcarriers. The 52 used subcarriers (data + pilot) are assigned numbers from -26,-25, …,-2,-1 and 1,2,…,25,26. The following figure illustrates the scheme of assigning these subcarriers to the IFFT inputs.



Matlab Code:

%Simulation of OFDM system in an AWGN environment
Follows IEEE 802.11 specification

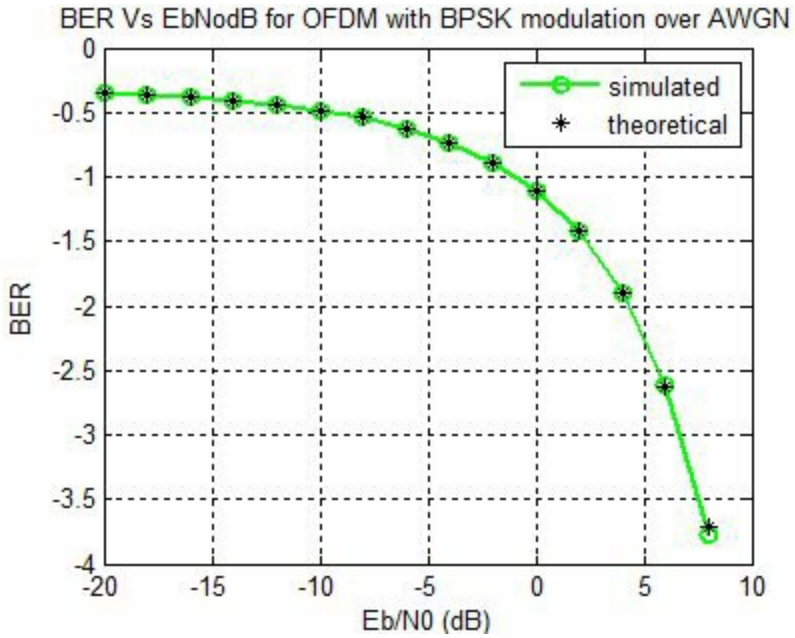
clear; clc;
%--------Simulation parameters----------------
nSym=10^4; %Number of OFDM Symbols to transmit

```
EbN0dB = -20:2:8; % bit to noise ratio
%-----------------------------------------

%--------OFDM Parameters - Given in IEEE Spec--
N=64; %FFT size or total number of subcarriers (used + unused) 64
Nsd = 48; %Number of data subcarriers 48
Nsp = 4 ; %Number of pilot subcarriers 4
ofdmBW = 20 * 10^6 ; % OFDM bandwidth
%-----------------------------------------

%--------Derived Parameters--------------------
deltaF = ofdmBW/N;  %=20 MHz/64 = 0.3125 MHz
Tfft = 1/deltaF; % IFFT/FFT period = 3.2us
Tgi = Tfft/4;%Guard interval duration - duration of cyclic prefix
Tsignal = Tgi+Tfft; %duration of BPSK-OFDM symbol
Ncp = N*Tgi/Tfft; %Number of symbols allocated to cyclic prefix
Nst = Nsd + Nsp;  %Number of total used subcarriers
nBitsPerSym=Nst; %For BPSK the number of Bits per Symbol is same as num of subcarriers

%-----------------------------------------
EsN0dB = EbN0dB + 10*log10(Nst/N) + 10*log10(N/(Ncp+N)); % converting to symbol to noise
ratio
errors= zeros(1,length(EsN0dB));
theoreticalBER = zeros(1,length(EsN0dB));

%Monte Carlo Simulation
for i=1:length(EsN0dB),
    for j=1:nSym
        %-----------------Transmitter--------------------
        s=2*round(rand(1,Nst))-1; %Generating Random Data with BPSK modulation
        %IFFT block
        %Assigning subcarriers from 1 to 26 (mapped to 1-26 of IFFT input)
        %and -26 to -1 (mapped to  38 to 63 of IFFT input); Nulls from 27 to 37
        %and at 0 position
        X_Freq=[zeros(1,1) s(1:Nst/2) zeros(1,11) s(Nst/2+1:end)];
        % Pretending the data to be in frequency domain and converting to time domain
        x_Time=N/sqrt(Nst)*ifft(X_Freq);

        %Adding Cyclic Prefix
        ofdm_signal=[x_Time(N-Ncp+1:N) x_Time];

        %--------------Channel Modeling ----------------
        noise=1/sqrt(2)*(randn(1,length(ofdm_signal))+1i*randn(1,length(ofdm_signal)));
        r= sqrt((N+Ncp)/N)*ofdm_signal + 10^(-EsN0dB(i)/20)*noise;
```

```
        %----------------Receiver----------------------
        %Removing cyclic prefix
        r_Parallel=r(Ncp+1:(N+Ncp));

        %FFT Block
        r_Time=sqrt(Nst)/N*(fft(r_Parallel));

        %Extracting the data carriers from the FFT output
        R_Freq=r_Time([(2:Nst/2+1) (Nst/2+13:Nst+12)]);

        %BPSK demodulation / Constellation Demapper.Force +ve value --> 1, -ve value --> -1
        R_Freq(R_Freq>0) = +1;
        R_Freq(R_Freq<0) = -1;
        s_cap=R_Freq;
        numErrors = sum(abs(s_cap-s)/2); %Count number of errors
        %Accumulate bit errors for all symbols transmitted
        errors(i)=errors(i)+numErrors;
    end
    theoreticalBER(i)=(1/2)*erfc(sqrt(10.^(EbN0dB(i)/10))); %Same as BER for BPSK over AWGN
end
simulatedBER = errors/(nSym*Nst);
plot(EbN0dB,log10(simulatedBER),'r-o');
hold on;
plot(EbN0dB,log10(theoreticalBER),'k*');
grid on;
title('BER Vs EbNodB for OFDM with BPSK modulation over AWGN');
xlabel('Eb/N0 (dB)');ylabel('BER');legend('simulated','theoretical');
```

Simulation Result:

From the following simulated result, it can be ascertained that the OFDM- BPSK modulation has no advantage over a normal BPSK system in AWGN. OFDM proves to be effective in multipath environments.



******

Back to Table of Contents

Chapter 8: Spread Spectrum Techniques

8.1 Introduction to Spread Spectrum Communication

Multiple Accesses:

Scenario: Multiple Users want to talk at the same time in the same geographical area

Example: 4 people want to talk at the same time in our locality using the same available resource (as in GSM)

Problem: How to allow different users to share the same resources?

Solutions: Divide them separately in time or in frequency or in combination of both or using separate codes

Dividing the users in Frequency Domain: – FDMA

FDMA – Frequency Domain Multiple Access is a technique in which the users share the same resources but at different frequencies.

Advantages of FDMA

* No Inter Symbol Interference (ISI)
* Simple to design
* Easier synchronization

Disadvantages of FDMA

Frequency reuse is a problem in FDMA. We cannot allocate indefinite number of frequencies to the

ever growing number of users. Since the spectrum is limited in bandwidth, a limited set of frequencies (Absolute Radio-Frequency Channel Number- ARFCN as in GSM) is reused over a given area. Need to take care of co-channel interference when using frequency reuse technique

High Q filters are required at the receiver side to separate the frequency content of each allocated spectrum, or large guard bands are required to separate each user.

Dividing the users in Time Domain: – TDMA

TDMA – Time Domain Multiple Access is a technique in which the users share the same resources but at different time slots.The time slots are so small that it is impossible for users to perceive the service disruptions.

Advantages of TDMA

* No need for High Q filters
* Suitable for digital data like PCM (Pulse coded modulation) data

Disadvantages of TDMA

* Need for tight synchronization circuits and guard time between time slots
* Equalizers are needed to cancel ISI effects.
* Susceptible to interference and multi-path effects.

Dividing the Users using codes: CDMA (Code Division Multiple Access)

Spread Spectrum Techniques:

The strict requirements of FDMA and TDMA are overcome by using spread spectrum techniques. Spread Spectrum techniques do not require high Q analog filters, frequency reuse, guard bands (as in FDMA) and equalizers, strict synchronization circuits, guard time (as in TDMA).

Advantages:

* Low power spectral density however the transmitted bandwidth requirement is higher than the other two techniques.
* Privacy due to use of separate codes for each users
* Possibility of jamming the signal is very low
* Reduced multi-path effects
* Immune to interference effects

There exist different techniques to spread a signal: Direct-Sequence (DS), Frequency-Hopping (FH), Time-Hopping (TH) and Multi-Carrier CDMA (MC-CDMA). It is also possible to make use of combinations of them.

Direct-Sequence (DS) spread spectrum:

In DS spread spectrum technique, the user data is spread over much higher bandwidth by multiplying with a maximum length PN sequence called chips. Always the user data rate (Rb) is relatively low compared to the rate of the PN sequence (called “chip rate” – Rc).The spread signal looks like a noise when transmitted over a radio interface.

“Spreading Factor” or “Processing Gain” determines number of users that can be allowed in a spread

spectrum system, the amount of multi-path effect reduction, the difficulty to jam or detect a signal etc. It is defined as the ratio of chip rate to the user data rate. Higher the spreading factor, greater is the capability to detect user data. Spreading factor may also be considered as a "gain" and is known as "Processing Gain" given by 10log10 (chip rate/user rate).

If the processing gain of a CDMA system is 20dB and if a Eb/N0 ratio of 5dB is needed at the receiver for satisfactory operation , the signal-to-interference ratio can be as low as -15dB, the user signal can still be recovered from the received signal (in presence of the interference signal). This is because the de-spreading benefits from the processing gain of 20dB.

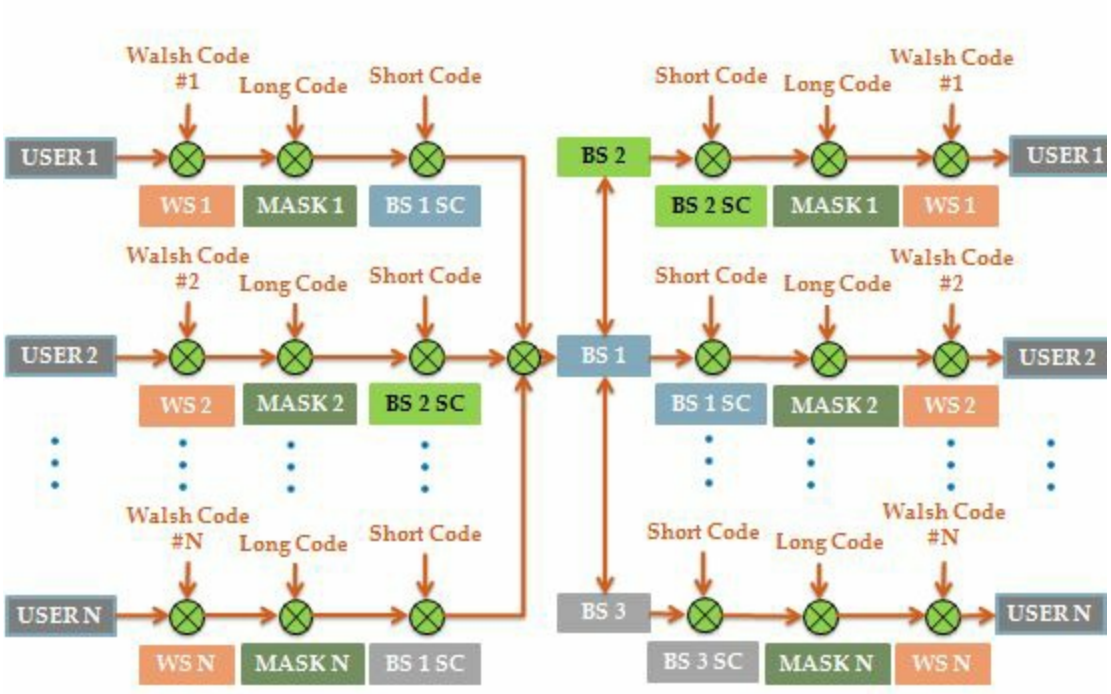
In a typical WCDMA system, the chip rate is 3.84×10^6 chips/second (3.84 Mcps) and the user data rate is 12.2 Kbps, which equates to a processing gain of 10log (3.84M/12.2K)=24.9dB.

******

Back to Table of Contents

8.2 Codes used in CDMA

IS-95 CDMA Standard uses three types of codes namely 1) Long code 2) Short code and 3) Walsh Hadamard codes.



Long Code:

Long codes run at 1.2288 Mb/s and are 42 bits longs (created using a 42 bit LFSR register). It takes approx. 41.2 days to repeat a long code at this rate. It is used for both encryption and spreading. Encryption is achieved by using a mask called Long Code mask which is a created using a 64-bit authentication key called A-key (assigned by CAVE protocol) and Electronic Serial Number (ESN – assigned each user based on the mobile number). The Long code changes each time a new connection is created.

Short Code:

Short code is an m-sequence of length 215-1 (created using a 15 bit LFSR register) and is used for synchronization purpose in the forward as well as reverse links. The short code is also used to identify cell/base station connection in the forward link. It repeats approx. 75 times in 2 seconds. Each base station is assigned a cyclically shifted version of same short code sequence to differentiate the base stations. This is also called PN offset in CDMA jargon. Since the cyclically shifted versions of the same m-sequence offer poor correlation, it is easier to differentiate between different base station links.

During the initial call setup stage, a mobile phone tries to find a base station (in 2 seconds max allowed time), if it find a base station, the mobile phone is validated using a database by the base station and is assigned a PN Short code sequence. This PN short code sequence uniquely identifies the connection between the particular base station and the mobile devices served under that base station.

In reality two short code sequences are used; one for I channel and another for Q channel (used in spreading and de-spreading).

Walsh Hadamard Code:

CDMA uses another type of code called Walsh Hadamard Code. In IS-95 CDMA, 64 Walsh codes are used per base station. This enables to create 64 separate channels per base stations (i.e. a base station can handle maximum 64 unique users at a given time). In CDMA-2000 standard, 256 Walsh codes are used to handle maximum 256 unique users under a base.

Walsh codes are created by transformation of Hadamard Matrix. The codes under a family of Walsh codes, possess a beautiful property of being orthogonal to each other (what else do we want to identify/accommodate users?).

The first matrix in a Hadamard transform is

$$H1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

The subsequent matrices are formed iteratively using

$$H_{N+1} = \begin{bmatrix} H_N & H_N \\ H_N & \overline{H}_N \end{bmatrix}$$

For example the H_2 matrix will be

$$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & \overline{H}_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Each row of a Hadamard matrix represents a unique Walsh code and all the Walsh codes in a given matrices are orthogonal. The length of the row of the matrix (number of columns otherwise) is the code-length of the Walsh codes. To get a 64-Walsh code matrix we need to transform the matrices till H8 (this matrix contains 64 rows – representing 64 Walsh codes and each code is of 64 bits length).

Walsh codes possess excellent cross-correlation property (cross correlation of one Walsh code with another is always zero) and therefore possess excellent orthogonality property. The auto-correlation property of Walsh code is very poor and so it is used only in synchronous CDMA networks, which maintains a synchronizing mechanism to identify the starting of the codeword.

Actually in IS-95, out of the 64 available Walsh codes, Walsh code 0 is reserved for pilot channel, 1 to 7 are assigned for synch channel and paging channels and the remaining 8-63 are assigned for users (traffic channel).

Matlab Code to simulate Walsh Hadamard Codes:

The following is a Matlab function to generate a Walsh Hadamard Matrix of given codeword size. The codeword size has to be a power of 2.

```
function [H]=generateHadamardMatrix(codeSize)

%[H]=generateHadamardMatrix(codeSize);
% Function to generate Walsh-Hadamard Matrix where "codeSize" is the code
% length of walsh code. The first matrix gives us two codes; 00, 01. The second matrix gives: 0000,
0101, 0011, 0110 and so on


N=2;
H=[0 0 ; 0 1];
if bitand(codeSize,codeSize-1)==0
    while(N~=codeSize)
        N=N*2;
        H=repmat(H,[2,2]);
        [m,n]=size(H);
        %Invert the matrix located at the bottom right hand corner
        for i=m/2+1:m,
          for j=n/2+1:n,
           H(i,j)=~H(i,j);
          end
        end
    end
    else
    disp('INVALID CODE SIZE:The code size must be a power of 2');
end
```

Example:

To Generate Walsh Codes used in IS-95 (which utilizes 64 Walsh codes of size 64 bits each, use : [H]=generateHadamardMatrix(64). This will generate 64 Walsh Codes of length 64-bits (for each code).

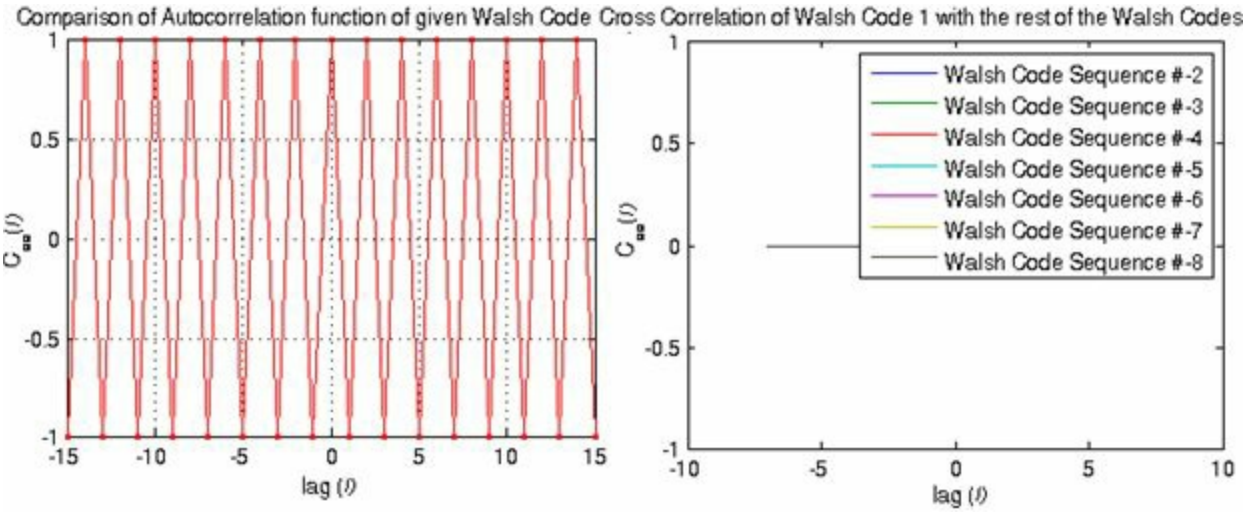
Test Program:

Also given below is a program to test the cross-correlation and auto-correlation of Walsh code. A set of 8-Walsh codes are used for this purpose.

```
% Matlab Program to test Walsh Hadamard Codes and to test their orthogonality
% Plots cross-correlation and auto correlation of Walsh Hadamard Codes
codeSize=8;

[H]=generateHadamardMatrix(codeSize);

%-------------------------------------------------------
%Cross-Correlation of Walsh Code 1 with rest of Walsh Codes
h = zeros(1, codeSize-1); %For dynamic Legends
s = cell(1, codeSize-1); %For dynamic Legends
for rows=2:codeSize
    [crossCorrelation,lags]=crossCorr(H(1,:),H(rows,:));
    h(rows-1)=plot(lags,crossCorrelation);
    s{rows-1} = sprintf('Walsh Code Sequence #-%d', rows);
    hold all;
end

%Dynamic Legends
% Select the plots to include in the legend
index = 1:codeSize-1;

% Create legend for the selected plots
legend(h(index),s{index});
title('Cross Correlation of Walsh Code 1 with the rest of the Walsh Codes');
ylabel('Cross Correlation');
xlabel('Lags');

%-------------------------------------------------------
%AutoCorrelation of Walsh Code - 1
autoCorr2(H(2,:),8,2,1);
```

Simulation Results

From the plots below, it can be ascertained that the Walsh codes have excellent cross-correlation property and poor autocorrelation property. Excellent cross-correlation property (zero cross-correlation) implies orthogonality, which makes it suitable for CDMA applications.



******

Back to Table of Contents

8.3 Maximum Length Sequences (m-sequences)

In spread spectrum communications, typically in CDMA, the user data is multiplied with a spreading sequence to achieve spreading. When the signal is received, the spreading is removed from the desired signal by multiplying it with the same sequence that was exactly synchronized to the transmitted PN signal. When a de-spreading operation is applied to the interferer's signals, it spreads the interference signal over a wider bandwidth, thereby reducing its effective power. This reduction in the interference power translates as processing gain.

In CDMA, each user is assigned a predetermined spreading sequence which has low cross correlation property with other user's spreading sequences.

Spreading sequences are chosen based on their characteristics like autocorrelation, cross correlation properties, etc… Some of the spreading sequences are listed below

1) Maximum length Pseudo Noise (PN) sequence
2) Gold sequences
3) Kasami sequences
4) Walsh Hadamard sequences

Maximum Length PN sequences (m-sequences):

Maximum Length PN sequence generators are binary sequence generators that are capable of outputting all possible combinations of binary sequences in 2^m-1 cyclic shifts, where m is the size of the LFSR (Linear Feedback Shift Registers) used in generating such sequences. Terms like Pseudo-Random Binary Sequences (PRBS) or pseudo-noise sequences are also used to refer m-sequences.

To generate an m-sequence, feedback connections of LFSRs are done according to a primitive polynomial (generator polynomial).

Primitive Polynomials:

A generator polynomial is said to be primitive if it cannot be factored (i.e. it is prime), and if it is a factor of (i.e. can evenly divide) x^N+1, where $N = 2^m-1$ (the length of the m-sequence).

For example, if we wish to construct an m-sequence generator using 3 registers (m=3), then the primitive polynomial that determines the feedback connection can be determined as follows.

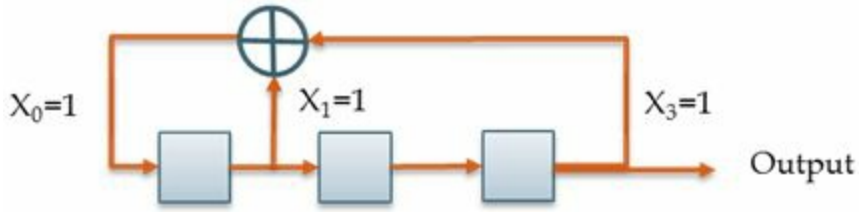
$$N = 2^m - 1$$
$$x^7 + 1 = (x+1)(x^3 + x + 1)(x^3 + x^2 + 1)$$

In the above equation, the three primitive polynomials are $(x+1)$, $(x^3 + x + 1)$ and $(x^3 + x^2 + 1)$.

Since the number of registers are m=3, we have to choose a primitive polynomial that is of degree 3. From the above factorized equation, we have two choices for m=3, i.e. $(x^3 + x + 1)$and $(x^3 + x^2 + 1)$ .These polynomials are often represented as [3 1 0] and [3 2 1] respectively. Some of the primitive polynomials used for generating m-sequences are given below.

Degree(m)	Length of m-Sequence (N)	Primitive Polynomials
1	1	$x+1$
2	3	x^2+x+1
3	7	x^3+x+1
4	15	x^4+x+1
5	31	x^5+x^2+1
6	63	x^6+x+1
7	127	x^7+x+1
8	255	$x^8+x^7+x^2+x+1$
9	511	x^9+x^4+1
10	1023	$x^{10}+x^3+1$
11	2047	$x^{11}+x^2+1$
12	4095	$x^{12}+x^6+x^4+x+1$

Following is an example of generating an m-sequence of length 7.Generator polynomial = [3 1 0] which can be represented as G= [1 0 1 1] (G=[X_3 X_2 X_1 X_0]).The LFSR feedback connections are connected as follows.



Autocorrelation Property of m-sequences:

The discrete autocorrelation of an m-sequence is computed by,

$$C_{aa}(l) = \frac{1}{N}\sum_{k=0}^{N-1} a_k a_{k-l}$$

Since m-sequences are periodic with period N, the autocorrelation is also periodic with period N.

$$C_{aa}(l) = C_{aa}(l+N)$$

Theoretically the autocorrelation of an m-sequence is a two valued function given by

$$C_{aa}(l) = \begin{cases} 1, & l = kN \\ \frac{1}{N}, & l \neq kN \end{cases} \quad ; k = 0,1,2,\ldots,N \quad 1$$

Matlab Code:

In the following Matlab code, a maximum length PN sequence is generated using a primitive polynomial of degree 6. The generator polynomial used is [6 1 0] . Since m=6, N=63 (2^m-1). Therefore, the generated m-sequence will be periodic with period N=63 i.e., all the values and autocorrelation repeats after 63 samples. An NRZ coder is used to convert the generated output to polar format.

The autocorrelation of the m-sequence is two valued: 1 and 0.0158 (because 1/N=1/63=0.0158). The Matlab code given below is generic and it can be used to generate any PN sequence (any primitive polynomial). If you want to run with a different polynomial, just change line 2 in main file (testPNSequence.m).

File1: testPNSequence.m

```
%testPNSequence.m
clc;
polynomial=[6 1 0]; %example: AX^3+BX^1+1 is represented as [A B 0]
[pnSequence,chipIndex,N]=genPNSequence(polynomial);
%Plot autocorrelation over 2 periods and doPlot
[autoCorrelation,lags]=autoCorr(pnSequence,N,2,1);
%Converting to Polar format
[time,yCoded,Fs]=NRZ_Encoder(y,1000,1,'Polar');
figure;
subplot(2,1,1);
stem([1:N],y,'-r');
title('Generated m-Sequence');
xlabel('Chip Index (n)');
ylabel('y');
ylim([-0.5 1.5]);grid on;
subplot(2,1,2);
plot(time,yCoded,'-');
title('Generated m-Sequence coded into Polar format');
xlabel('Time (t)');
ylabel('yCoded');
ylim([-1.5 1.5]); grid on;
```

File2: genPNSequence.m : PN Sequence Generator

```
function [pnSequence,chipIndex,N]=genPNSequence(polynomial)
%function to generate a maximal length PN sequence (m-sequence)
%Argument : Examples for polynomial input is as follows
%Polynomial [X^3+X^1+1] is given as [3 1 0]
```

```
%Polynomial [X^6+X^1+1] is given as [6 1 0]
%Generates the PN sequence chip Index and the length of the Sequence N
%polynomial=[6 1 0]; %example: AX^3+BX^1+1 is represented as [X3 X2 X1 X0]->;[A B 0]
N=2^polynomial(1)-1;
%convert polynomial representation to BnX^n+Bn-1x^n-1+...B1X^1+B0X^0 form
temp=zeros(1,max(polynomial)+1);
for i=1:length(polynomial)
    temp(length(temp)-polynomial(i))=1;
end
polynomial=temp;
%initialize the initial state of the LFSR
x=[1 zeros(1,length(polynomial)-2)];
y=zeros(1,N);
for i=1:N
    y(i)=x(end); %output;
    xi=mod(sum(x.*polynomial(2:end)),2);
    x(2:end)=x(1:end-1);
    x(1)=xi;
end
pnSequence = y;
chipIndex = 1:N;
```

File 3: autoCorr.m: Computes the autocorrelation of m-sequence

```
%autoCorr.m
%Computes autocorrelation function of a sequence
function [autoCorrelation,lags]=autoCorr(sequence,N,Np,doPlot)
%Accepts four arguments namely "sequence",N,Np and "doPlot" sequence is the input sequence (1s
and 0s) of which the autocorrelation is desired
% N is the length of the m sequence
% Np is the number of periods over which is autocorrelation is desired
%if doPlot = 1, the results of the autocorrelation will be plotted
%if doPlot = 0, plots will be ignored.
if nargin==1
    doPlot=0;
end

%Covert to polar form
a = 1 - 2*sequence;
a_present = repmat(a,1,Np); %copy the sequence Np times
L = length(a_present);
%Compute the autocorrelation
a_past = a_present'; %delay=0

for k = 1:L
    C(k) = (a_present*a_past)/L;
```

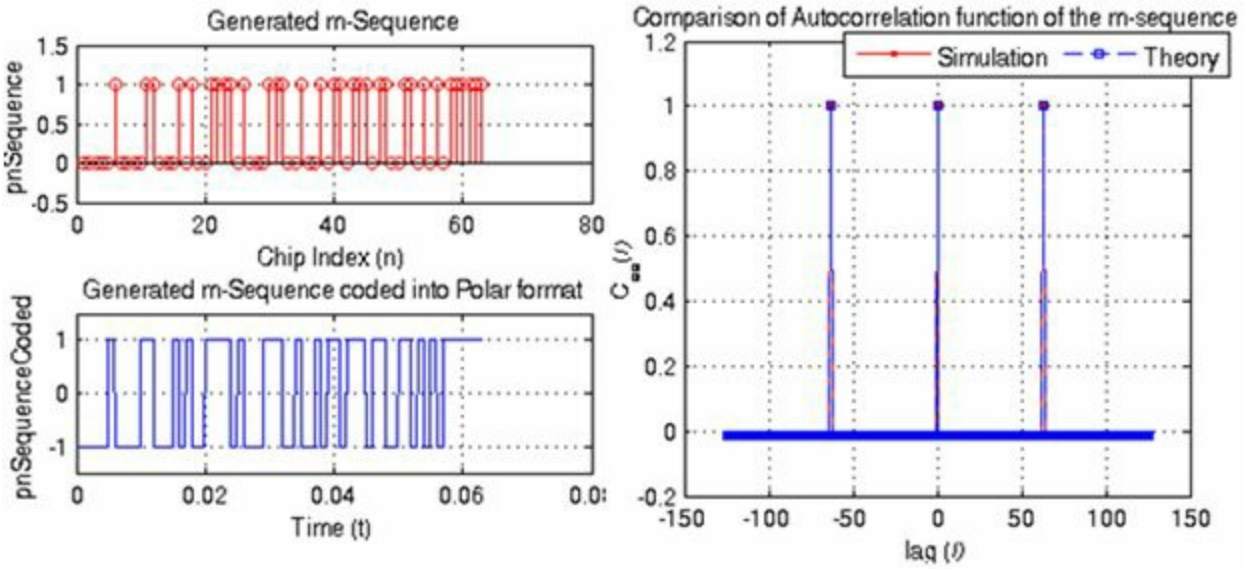
```
    a_past_out = a_past(end);
    a_past(2:end) = a_past(1:end-1);
    a_past(1) = a_past_out;
end
%Computing the simulated Autocorrelation using Conjugate Symmmetry property
C_sim = [conj(fliplr(C(2:end))) C];
%delay vector
l = -(L-1):(L-1);
m = -(Np-1):(Np-1);

%Compute theoretical Autocorrelation
C_theory = (-1/N)*ones(2*L-1,1);
for i=1:length(m)
if (L+m(i)*N>0) && (L+m(i)*N)
    C_theory(L + m(i)*N) = 1;
    end
end
if doPlot
    figure;
    plot(l,C_sim,'-*r','markersize',3);
    hold on;
    plot(l,C_theory,'--s','markersize',4);
    hold off;
    legend('Simulation','Theory','orientation','horizontal');
    title('Comparison of Autocorrelation function of the m-sequence');
    xlabel('lag (\it{l})');
    ylabel('C_{aa}(\it{l})');
    grid;
end
autoCorrelation=C_sim;
lags=l;
```

File 4: NRZ_Encoder.m : NRZ encoder to convert to polar form

```
%NRZ_Encoder.m
% NRZ_Encoder   Line codes encoder
%[time,output,Fs]=NRZ_Encoder(input,Rb,amplitude,style)
%NRZ_Encoder
%Input a stream of bits and specify bit-Rate, amplitude of the output signal and the style of encoding
%Currently 3 encoding styles are supported namely 'Manchester','Unipolar'and 'Polar'
%Outputs the NRZ stream
function [time,output,Fs]=NRZ_Encoder(input,Rb,amplitude,style)
Fs=10*Rb; %Sampling frequency , oversampling factor= 32
Ts=1/Fs; % Sampling Period
Tb=1/Rb; % Bit period
output=[];
```

```
switch lower(style)
    case {'manchester'}
        for count=1:length(input),
          for tempTime=0:Ts:Tb/2-Ts,
           output=[output (-1)^(input(count))*amplitude];
          end
          for tempTime=Tb/2:Ts:Tb-Ts,
           output=[output (-1)^(input(count)+1)*amplitude];
          end
        end
    case {'unipolar'}
        for count=1:length(input),
          for tempTime=0:Ts:Tb-Ts,
           output=[output input(count)*amplitude];
          end
        end
    case {'polar'}
        for count=1:length(input),
          for tempTime=0:Ts:Tb-Ts,
           output=[output amplitude*(-1)^(1+input(count))];
          end
        end
    otherwise,
        disp('NRZ_Encoder(input,Rb,amplitude,style)-Unknown method given as "style" argument');
        disp('Accepted Styles are "Manchester", "Unipolar" and "Polar"');
end
time=0:Ts:Tb*length(input)-Ts;
```

Simulation Results:



******

Back to Table of Contents

8.4 Preferred Pairs m-sequences generation for Gold Codes

Spreading sequences in spread spectrum systems can be generated with help of diversified codes like m-sequences, Gold Codes, Kasami Codes, Walsh Codes etc.

Compared to m-sequences (maximum length PN Sequences), Gold codes have worst auto-correlation properties but they have better cross-correlation properties.

The sequences associated with Gold Codes are produced by binary addition (modulo-2) of cyclic shifted versions of two m-sequence of length $N=2^n-1$. If two m-sequences of cyclic shift t_1 and t_2 are used in Gold code generation, then the generated gold code will be unique for each combination of t_1 and t_2. This means that plenty of gold codes are generated with just two m-sequences which implies larger number of users can be accommodated in a given spread spectrum system.

When the two m-sequences are picked randomly for Gold code generation, then the cross-correlation property of the generated Gold code might not be as good as expected. Gold codes are generated using "Preferred" pairs of sequences that will guarantee good cross-correlation (as well as auto-correlation) properties of the generated Gold code. A method for selecting the preferred pairs for Gold Code generation was given by Gold [GoldCode] and is detailed here.

Generating Preferred Pairs of m-sequences:

1) Take an m-sequence (d) for given length N ($N = 2^n - 1$, where n is the number of registers in the LFSR).

2) Decimate the m-sequence by a decimation factor of q. This is our second sequence. $d' = d[q]$

3) If the value of q is chosen according to the following three conditions, then the two m-sequences d and d' will be preferred pairs.

Condition A) n is odd or mod (n, 4) =2

Condition B) q is odd and either $q = 2^k + 1$ or $q = 2^{2k} - 2k + 1$ for an integer k.

Condition C) The greatest common divisor of n and k, gcd (n, k) =1 when n is odd and gcd (n, k) =2 when mod (n, 4) =2

Example:

Let us consider the generation of a preferred pair of m-sequences of length N=63 (n=6). Since mod(n=6,4)=2, the first condition is satisfied. Taking q=5 and k=2 satisfies condition 2 and 3. So we will use a decimation factor of q=5 in our simulation for preferred pairs generation.

Cross-Correlation:

The cross-correlation property of the preferred pairs for Gold code generation is three valued and the values are,

$$\left[-\frac{1}{N}t(n), -\frac{1}{N}, \frac{1}{N}(t(n)-2)\right]$$

where t(n) is given by

$$t(n) = \begin{cases} 2^{(n+1)/2}+1 & ;if\ n\ is\ odd \\ 2^{(n+2)/2}+1 & ;if\ n\ is\ even \end{cases}$$

For this example, the theoretical cross-correlation values for the preferred pairs of m-sequences are [-0.2698, -0.0159, 0.2381].

Matlab Code:

File 1: preferredPairs.m

```
%Main Program - Execute this
clc; clear all;
n=6;
N=2^6-1;
if rem(n,2) || (mod(n,4)==2)
    disp('Condition 1 Satisfied');
end
q=5;
[d,dDecimated]=genDecimatedPNSequence([6 1 0],q);

figure;
subplot(2,1,1);
plot(d,'-sr','markersize',4);
title('Preferred Pair of m-sequences');
ylabel('PN Sequence - 1');
xlabel('Chip Index (k)');
ylim([-0.2 1.2]);
grid on;

subplot(2,1,2);
plot(dDecimated,'-*k','markersize',4);
title('Preferred Pair of m-sequences');
ylabel('PN Sequence - 2');
xlabel('Chip Index (k)');
ylim([-0.2 1.2]);
grid on;

% simulated Cross Correlation
[C_sim, lags] =crossCorr (d, dDecimated);
figure;
plot(lags,C_sim,'-*r','markersize',4);
title('Cross Correlation of Preferred Pair of m-sequences');
ylabel('C_sim');
```

```
xlabel('Lag (l)');
grid on;

%Theoretical Cross Correlation
if rem(n,2)
    tn=1+2^(0.5*(n+1)); %For n=odd
else
    tn=1+2^(0.5*(n+2)); %For n=even
end
theoreticalCorr=[-1/N*tn,-1/N,1/N*(tn-2)]
```

File2: genPNSequence.m

```
%genPNSequence.m

function [pnSequence,chipIndex,N]=genPNSequence(polynomial,initialCondition)
%function to generate a maximal length PN sequence (m-sequence)
%Argument : Examples for polynomial input is as follows
%Polynomial [X^3+X^1+1] is given as [3 1 0]
%Polynomial [X^6+X^1+1] is given as [6 1 0]
%Generates the PN sequence,chip Index and the length of the Sequence N
%polynomial=[6 1 0]; %example: AX^3+BX^1+1 is represented as [X3 X2 X1 X0]->[A B 0]
N=2^polynomial(1)-1;
%convert polynomial representation to BnX^n+Bn-1x^n-1+...B1X^1+B0X^0 form
temp=zeros(1,max(polynomial)+1);
for i=1:length(polynomial)
    temp(length(temp)-polynomial(i))=1;
end
polynomial=temp;

if nargin<2
    initialCondition=[1 zeros(1,length(polynomial)-2)];
end
%initialize the initial state of the LFSR x=initialCondition;
y=zeros(1,N);

for i=1:N
    y(i)=x(end); %output;
    xi=mod(sum(x.*polynomial(2:end)),2);
    x(2:end)=x(1:end-1);
    x(1)=xi;
end
pnSequence = y;
chipIndex = 1:N;
```

File3 : genDecimatedPNSequence.m

```
%genDecimatedPNSequence.m
```

Function

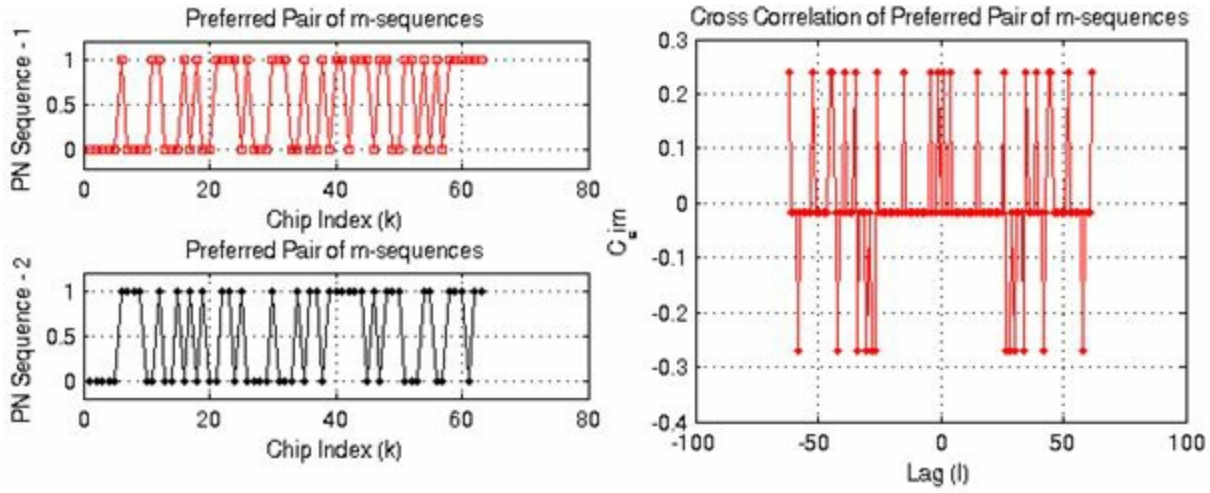
```
[pnSequence,decimatedVersion]=genDecimatedPNSequence(polynomial,decimationFactor)
%function to generate a decimated maximal length PN sequence (m-sequence)
q=decimationFactor;
pnSequence=genPNSequence(polynomial);
dRep=repmat(pnSequence,1,q);
decimatedVersion=dRep(1,q:q:end);
```

File4: crossCorr.m

```
%crossCorr.m
%Computes crosscorrelation function of two sequences
function [crossCorrelation,lags]=crossCorr(sequence1,sequence2)

%Covert to polar form
a1 = 1 - 2*sequence1;
a2 = 1 - 2*sequence2;
a1_present = a1;
a2_present = a2;
L = length(a1_present);

%Compute the cross correlation
a_past = a2_present'; %delay=0
a_present=a1_present;
for k = 1:L
    C(k) = (a_present*a_past)/L;
    a_past_out = a_past(end);
    a_past(2:end) = a_past(1:end-1);
    a_past(1) = a_past_out;
end

%Computing the simulated Cross correlation using Conjugate Symmmetry property
C_sim = [conj(fliplr(C(2:end))) C];
%delay vector
l = -(L-1):(L-1);
crossCorrelation=C_sim;
lags=l;
```

Results:

The theoretical and simulated cross-correlation values for the generated preferred pairs (with N=63 and q=5) matches perfectly.



******

Back to Table of Contents

8.5 Generation of Gold Codes and their cross-correlation

In the previous section, generation of preferred pairs of m-sequences for gold codes was discussed. Let us see how to generate Gold sequences using the preferred pair of m-sequences (say d_1 and d_2).

As mentioned in the previous section, numerous Gold sequences can be generated just by time shifting two m-sequences that are preferred pairs and XORing them together.

If d_1 and d_2 are preferred pair m-sequences then a SET of Gold sequences can be represented as,

where T represents cyclic shift by 1 bit, T^2 represents cyclic shift by 2 bits etc..,

$$Gold(d_1, d_2) = \{d_1, d_2 d_1 + d_2, d_1 + T d_2, d_1 + T^2 d_2, \dots, d_1 + T^{N-1} d_2\}$$

Theoretical Cross-correlation of Gold sequences is three valued and is given by

$$\left[-\frac{1}{N}t(n), -\frac{1}{N}, \frac{1}{N}(t(n) - 2)\right]$$

where t(n) is given by

$$t(n) = \begin{cases} 2^{(n+1)/2} + 1 & ;if\ n\ is\ odd \\ 2^{(n+2)/2} + 1 & ;if\ n\ is\ even \end{cases}$$

Matlab Code:

The following Matlab Code generates a set of 6 Gold Sequences from a "preferred pair" of m-sequences (d_1, d_2). The first two sequences in the set are d1 and d2 itself (obtained by zero cyclic shifts). Next 4 sequences are obtained by xoring d_1 with cyclic shifted versions of d_2.

Finally, Gold sequence 1 is cross correlated with the rest of the sequences and the simulated values are plotted. From the plot it is evident that the simulated cross-correlation values match with the theoretical values {-0.2698, 0.0159, 0.2381}.

Matlab Code:

File1: genGoldCodes.m

```
%Main Program
%Generated Gold Codes by generating two m-sequences d1 and d2.
%Here d2 is the decimated version of d1. For good correlation properties,
%d1 and d2 must be preferred pairs (based on decimation factor q and register size n).
%A set of gold sequence contains original d1 and d2 and XORed versions of cyclic shifted versions of d1 and d2, depicted as
%GoldSequences(d1,d2)=[d1,d2,d1+Td2,d1+T^2d2,d1+T^3d2,...d1+T^(N-1)d2]
clc; clear all;
n=6; %Register size for m-sequence generation
polynomial = [6 1 0]; %Polynomial m-sequence
numOfGoldSeq=6; %Number of Gold sequences to generate
q=5; %Decimation factor for preffered pair generation
%Be careful in chosing q, inappropriate value for q may result in poor cross-correlation valued Gold codes
N=2^n-1; %Length of m-sequence

%d and dDecimated are preferred pair (if and only if the three conditions for preferred-pair generation are satisfied)

%Generate Preferred pairs by decimation
[d,dDecimated]=genDecimatedPNSequence([6 1 0],q);
%Generate Gold code from Preferred Pairs
goldSequences = zeros(2+numOfGoldSeq,N);
goldSequences(1,:)=d;
goldSequences(2,:)=dDecimated;
dDecimatedShifted=dDecimated;

for rows = 3:2+numOfGoldSeq
    goldSequences(rows,:)=xor(d,dDecimatedShifted);
    %Cyclic Shifting dDecimated by 1 bit for each Gold Sequence
    dTemp=dDecimatedShifted(end);
    dDecimatedShifted(2:end) = dDecimatedShifted(1:end-1);
    dDecimatedShifted(1)=dTemp;
end

%Cross-Correlation of Gold Sequence 1 with rest of Gold Sequence
h = zeros(1, numOfGoldSeq-1); %For dynamic Legends
s = cell(1, numOfGoldSeq-1); %For dynamic Legends
for rows=2:2+numOfGoldSeq
    [crossCorrelation,lags]=crossCorr(goldSequences(1,:),goldSequences(rows,:));
    h(rows-1)=plot(lags,crossCorrelation);
    s{rows-1} = sprintf('Sequence-%d', rows);
```

```
    hold all;
end

%Dynamic Legends
% Select the plots to include in the legend
index = 1:numOfGoldSeq-1;

% Create legend for the selected plots
legend(h(index),s{index});
title('Cross Correlation of Gold Sequence 1 with other Gold Sequences');
ylabel('Cross Correlation');
xlabel('Lags');
ylim([-0.3 0.3]);
```

File2:genDecimatedPNSequence.m

```
%genDecimatedPNSequence.m

function

[pnSequence,decimatedVersion]=genDecimatedPNSequence(polynomial,decimationFactor)
%function to generate a decimated maximal length PN sequence (m-sequence)
q=decimationFactor;
pnSequence=genPNSequence(polynomial);
dRep=repmat(pnSequence,1,q);
decimatedVersion=dRep(1,q:q:end);
```

File3:genPNSequence.m

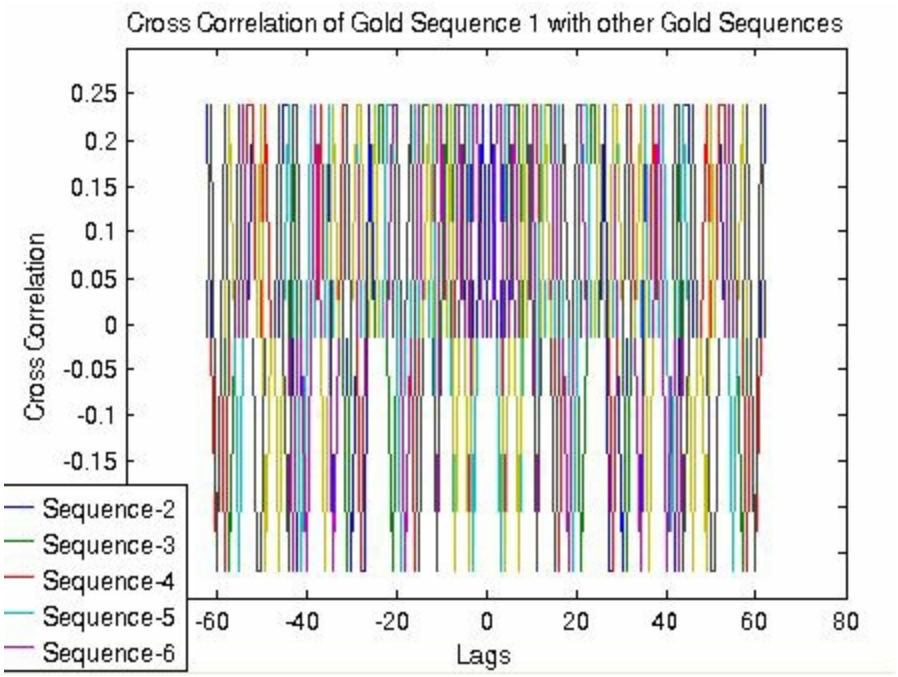
```
%genPNSequence.m

function [pnSequence,chipIndex,N]=genPNSequence(polynomial,initialCondition)
%function to generate a maximal length PN sequence (m-sequence)
%Argument : Examples for polynomial input is as follows
%Polynomial [X^3+X^1+1] is given as [3 1 0]
%Polynomial [X^6+X^1+1] is given as [6 1 0]
%Generates the PN sequence,chip Index and the length of the Sequence N
%polynomial=[6 1 0]; %example: AX^3+BX^1+1 is represented as [X3 X2 X1 X0]->[A B 0]
N=2^polynomial(1)-1;

%convert polynomial representation to BnX^n+Bn-1x^n-1+...B1X^1+B0X^0 form
temp=zeros(1,max(polynomial)+1);
for i=1:length(polynomial)
    temp(length(temp)-polynomial(i))=1;
end
polynomial=temp;

if nargin<2
    initialCondition=[1 zeros(1,length(polynomial)-2)];
```

```
end

%initialize the initial state of the LFSR
x=initialCondition;
y=zeros(1,N);
for i=1:N
    y(i)=x(end); %output;
    xi=mod(sum(x.*polynomial(2:end)),2);
    x(2:end)=x(1:end-1);
    x(1)=xi;
end
pnSequence = y;
chipIndex = 1:N;
```

File4: crossCorr.m

```
%crossCorr.m

%Computes crosscorrelation function of two sequences
function [crossCorrelation,lags]=crossCorr(sequence1,sequence2)
%Covert to polar form
a1 = 1 - 2*sequence1;
a2 = 1 - 2*sequence2;
a1_present = a1;
a2_present = a2;
L = length(a1_present);

%Compute the cross correlation
a_past = a2_present'; %delay=0
a_present=a1_present;

for k = 1:L
    C(k) = (a_present*a_past)/L;
    a_past_out = a_past(end);
    a_past(2:end) = a_past(1:end-1);
    a_past(1) = a_past_out;
end
%Computing the simulated Cross correlation using Conjugate Symmmetry property
C_sim = [conj(fliplr(C(2:end))) C];

%delay vector
l = -(L-1):(L-1);
crossCorrelation=C_sim;
lags=l;
```

Simulation Result:



*****

Back to Table of Contents

Appendix

A1: Deriving Shannon-Hartley Equation for CCMC AWGN channel -Method 1

Channel capacity is limited by two phenomenons: 1) Noise and 2) the input power constraint. An infinite capacity can be achieved for the following two scenarios:

1) The variance of noise is zero. This implies that for a constrained input power, the Shannon's limit approaches infinite if the noise variance in the channel is zero.

2) There is no limit to the power that is input to the channel (Transmission power is not limited).This follows from the fact that if we can transmit with infinite number of signaling points that are infinitely far apart, the channel will have infinite capacity no matter what the amount of noise is present in the channel. This implies that infinite capacity can be achieved for a fixed noise variance, by increasing the transmission power as we desire.

Consider a transmission system represented by the following figure with the input and output treated as random variables.

It is obvious that in order to have a meaningful expression for channel capacity over AWGN we place a constraint on the input power as follows

$$E[X^2] \leq S$$

Where

$$E[X^2] = \int_{-\infty}^{\infty} xp(x)dx$$

X=input random variable, p(x) = probability distribution of the unconstrained input X and $E[X^2]$ = second central moment of X.

Now the goal is to find the distribution of the input X, that satisfies the above mentioned constraint and maximizes the capacity at the same time.

Given an input constraint for all possible distributions of X, the capacity of a channel is defined as the amount of mutual information between the input X and the channel output Y – I(X;Y).

$$C = \max_{p(x):\, E[X^2] \leq S} I(X;Y) \qquad [1]$$

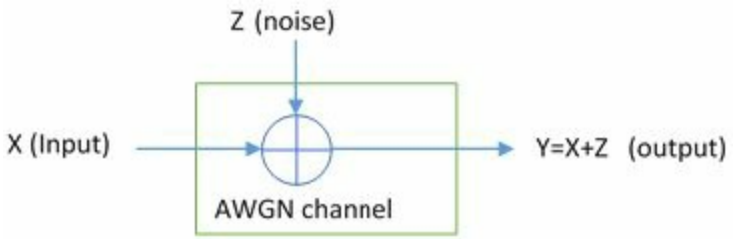
The mutual information is defined in terms of entropy function as

$$I(X;Y) = H(Y) \quad H(Y|X) \qquad [2]$$

Where, H(Y) is the entropy of channel output. It measures the amount of "uncertainty" of Y, H(Y|X) is the conditional entropy of the channel output given that we know the input value X.

The above equation can be viewed as the measure of reduction in the uncertainty of the channel output when we have the knowledge of the input value X.

Consider a communication system with AWGN channel as shown below.



For this given scenario, the mutual information in equation [2] is calculated as

$$I(X;Y) = H(Y) - H[(X+Z)|X]$$

Since the input X and the noise Z are statistically independent, the above equation can be re-written as follows

$$I(X;Y) - H(Y) - [H(X|X)] + H(Z|X)]$$

Since there is no uncertainty about determining 'X' given the knowledge of X, H(X|X)=0. Since the noise Z and the input X are completely independent, knowing X will not help in determining the knowledge of Z – H(Z|X) = H(Z). Thus the above equation shrinks to

$$I(X;Y) = H(Y) - H(Z) \qquad [3]$$

Given a Gaussian noise random variable Z~(0, σ^2) with the pdf,

$$f_z(z) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{\frac{-z^2}{2\sigma^2}\right\}$$

The continuous entropy of the random variable Z can be computed as follows,

$$H(Z) = -\int_{-\infty}^{\infty} f_z(z)\log_2\{f_z(z)\}\,dz$$

$$= -\int_{-\infty}^{\infty} f_z(z)\left(\log_2\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{z^2}{2\sigma^2}\log_2 e\right)dz$$

$$= \frac{1}{2}\log_2(2\pi\sigma^2) + \frac{\log_2 e}{2\sigma^2}E[Z^2] = \frac{1}{2}\log_2(2\pi e\sigma^2) \qquad [4]$$

Since we have no control over the entropy of noise H(z), it follows from equations [1] and [3] that we can conclude that the information capacity can be maximized if the entropy of output is maximized. Continuous Gaussian distribution maximizes the entropy. Thus if we can force the output entropy to be following a Gaussian distribution, then we have maximized the capacity. How do we force the output entropy to follow a Gaussian distribution? For an AWGN channel the output random variable Y is expressed as

$$Y = X + Z$$

Remember - that a sum of two Gaussian random variables will yield another Gaussian random variable. Thus if we make the input distribution to be a continuous Gaussian random variable, then the output distribution will also be continuous and Gaussian. Since the input is constrained with power – 'S', the second moment of the random variable Y is given by

$$E[Y^2] = E[Y^2] + E[Z^2]$$
$$E[Y^2] - S + \sigma^2$$

The expression for the output entropy is similar to the expression for the entropy of noise (equation 4), except for the term involving the variance. Thus, the output entropy is given by

$$H(Y) = \frac{1}{2}\log_2\{2\pi e(S+\sigma^2)\} \qquad [5]$$

Substituiting [4] and [5] in [3]

$$I(X;Y) = H(Y) \quad H(Z) = \frac{1}{2}\log_2\left(\frac{S+\sigma^2}{\sigma^2}\right) = \frac{1}{2}\log_2\left(1 + \frac{S}{\sigma^2}\right)$$

Thus the information capacity for the Gaussian Channel is

$$C = \max_{p(x):E[X^2]\le S} I(X;Y) = \frac{1}{2}\log_2\left(1 + \frac{S}{\sigma^2}\right)$$

The noise variance σ^2 can be substituted with noise power denoted 'N' and we get the equation in the most widely used form

$$C = \frac{1}{2}\log_2\left(1 + \frac{S}{N}\right)$$

The above mentioned capacity of the channel is in bits per channel usage. This capacity is defined for one sided unity bandwidth. Actually the bandwidth spans both positive and negative frequencies [-1Hz to +1Hz]. Thus the channel capacity for two sided bandwidth is given by

$$C = \log_2\left(1 + \frac{S}{N}\right)$$

If the given channel is band-limited from –B to B, then the capacity becomes

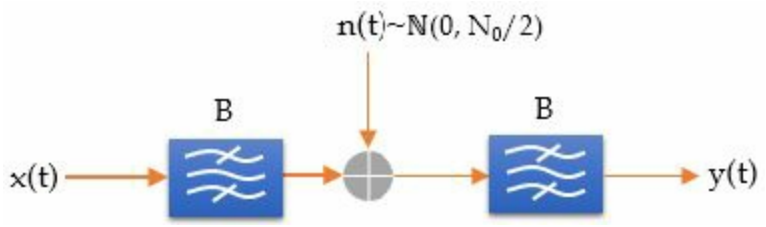
$$C = \mathrm{B}\log_2\left(1+\frac{S}{N}\right)$$

*****

Back to Table of Contents

A2. Capacity of Continuous input Continuous output Memoryless AWGN - Method 2

The generic capacity equation for CCMC channel (mentioned in chapter 1.5) is used here to find the capacity of an AWGN CCMC system. The pdf of an AWGN channel is Gaussian distributed. The following figure represents a CCMC AWGN band-limited channel of bandwidth B.



The capacity is calculated by driving the input with an independent-identically-distributed Gaussian noise source.

Let us drive the system with a white noise input (x_n). Assume that the input x_n is zero-mean Gaussian distributed with variance = σ^2 .Then, the pdf of N-dimensional input is given by

$$P(\bar{x}) = \prod_{n=1}^{N} P(x_n)$$

where,

$$P(x_n) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{\frac{-x_n{}^2}{2\sigma^2}\right\}$$

Also assume that the AWGN channel is zero-mean Gaussian with variance=$N_0/2$.The transition probability of the channel is given by

$$P(\bar{y}|\bar{x}) = \prod_{n=1}^{N} P(y_n|x_n)$$

where,

$$P(y_n|x_n) = \frac{1}{\sqrt{\pi N_0}} exp\left\{\frac{-(y_n - x_n)^2}{N_0}\right\}$$

The output of the channel is given by the sum of input and noise - both of which are Gaussian distributed. The pdf of the output is also Gaussian distributed with zero-mean and variance=$N_0/2+ \sigma^2$.

Therefore,

$$P(\bar{y}) = \prod_{n=1}^{N} P(y_n)$$

Where,

$$P(y_n) = \frac{1}{\sqrt{\pi(N_0 + 2\sigma^2)}} exp\left\{\frac{-{y_n}^2}{N_0 + 2\sigma^2}\right\}$$

Rewriting the capacity equation for the generic CCMC channel,

$$C = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P(\bar{x},\bar{y}) \log_2[P(\bar{y}|\bar{x})]\, d\bar{x}d\bar{y} - \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P(\bar{x},\bar{y}) \log_2[P(\bar{y})]\, d\bar{x}d\bar{y}$$
$$= A_1 - A_2$$

Solving for A_1,

$$\begin{aligned} A_1 &= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P(\bar{x},\bar{y}) \log_2[P(\bar{y}|\bar{x})]\, d\bar{x}d\bar{y} \\ &= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P(\bar{x},\bar{y}) \log_2 \left[\prod_{n=1}^{N} P(y_n|x_n)\right] d\bar{x}d\bar{y} \\ &= \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P(\bar{x},\bar{y}) \left\{\sum_{n=1}^{N} \log_2[P(y_n|x_n)]\, d\bar{x}d\bar{y}\right\} \\ &= \sum_{n=1}^{N} \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} P(\bar{x},\bar{y}) \log_2[P(y_n|x_n)]\, d\bar{x}d\bar{y} \end{aligned}$$

Note that A_1 is a sum of N-2N-fold integrals.Each double integral is for a single combination of y_n and x_n. Simplifying A_1,

$$\begin{aligned} A_1 &= N \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P(x,y) \log_2[P(y|x)]\, dxdy \\ &= N \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P(x,y) \log_2 \left[\frac{1}{\sqrt{\pi N_0}} exp\left\{\frac{-(y-x)^2}{N_0}\right\}\right] dxdy \\ &= N \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P(x,y) \left[\log_2\left[\frac{1}{\sqrt{\pi N_0}}\right] - \left\{\frac{(y-x)^2}{N_0}\right\}.\log_2 e\right] dxdy \\ &= N \log_2\left[\frac{1}{\sqrt{\pi N_0}}\right] \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P(x,y) dxdy - \frac{N \log_2 e}{N_0} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (y-x)^2 P(x,y) dxdy \\ &= N \log_2\left[\frac{1}{\sqrt{\pi N_0}}\right] - \frac{N \log_2 e}{N_0} \int_{-\infty}^{\infty} p(x) \int_{-\infty}^{\infty} P(y|x)(y-x)^2 dxdy \end{aligned}$$

Noting that the term $\int_{-\infty}^{\infty} P(y|x)(y-x)^2 dxdy$ is the second central moment of P(y|x) which is nothing but the variance of P(y|x),

$$A_1 = N\log_2\left[\frac{1}{\sqrt{\pi N_0}}\right] - \frac{N\log_2 e}{N_0}\cdot\frac{N_0}{2}\int_{-\infty}^{\infty} p(x)dxdy$$
$$= -\frac{N}{2}\log_2[\pi N_0] - \frac{N\log_2 e}{2}$$
$$= -\frac{N}{2}\log_2[e\pi N_0]$$

Deriving in a similar fashion, the value for A_2 is given by

$$A_2 = -\frac{N}{2}\log_2[e\pi(2\sigma^2 + N_0)]$$

Thus the capacity of the AWGN CCMC channel model is given by

$$C = -\frac{N}{2}\log_2[e\pi N_0] + \frac{N}{2}\log_2[e\pi(2\sigma^2 + N_0)]$$
$$= \frac{N}{2}\log_2\left(1 + \frac{2\sigma^2}{N_0}\right)$$

Since the input symbol is band-limited to B Hz and is time limited to T second, the dimensionality of the transmission is given by

$$N = 2BT$$

The average transmitted power is given by

$$P = \frac{1}{T}\sum_{n=1}^{N} E[x_n^2] = \frac{N\sigma^2}{T} = 2B\sigma^2$$

Therefore, the capacity equation can be written as

$$C = BT\log_2\left(1 + \frac{P}{BN_0}\right) \quad (bits/symbol)$$

When normalized to unit time, the capacity in bits/symbols/second is

$$C = B\log_2\left(1 + \frac{P}{BN_0}\right) \quad (bits/symbol/second)$$

The Signal to Noise Ratio (SNR) is defined as

$$SNR = \frac{P}{BN_0}$$

Thus the capacity in bps in terms of SNR is given by

$$C = B\log_2(1 + SNR) \quad (bits/symbol/second)$$

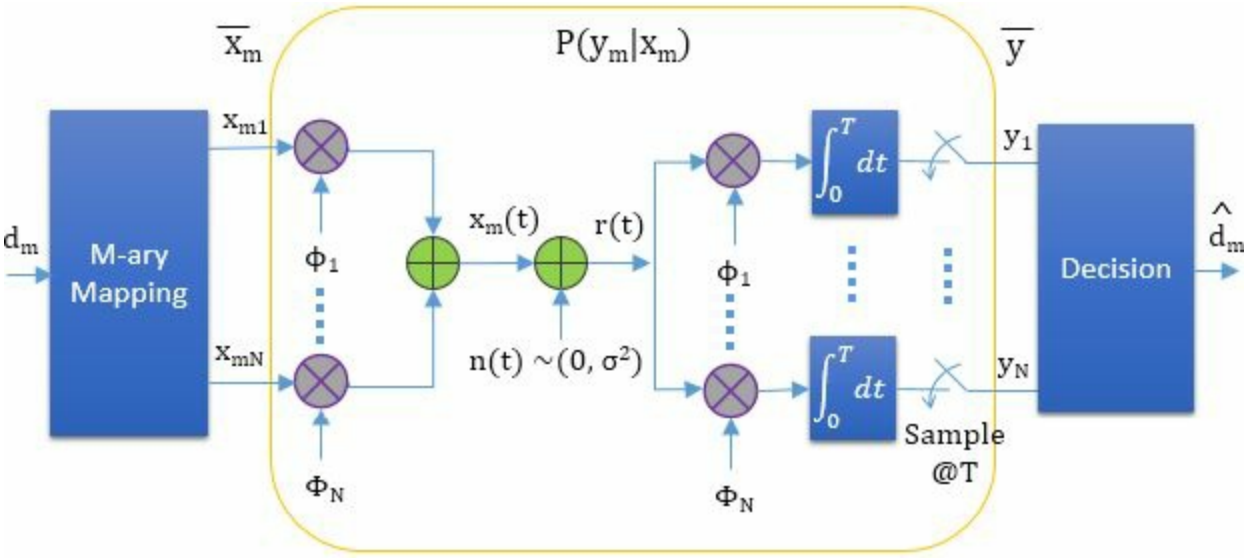
*****

Back to Table of Contents

A3: Constellation Constrained Capacity of M-ary Scheme for AWGN channel

The generic equation for DCMC channel given in chapter 1.5 is extended to a case where the channel is AWGN with zero mean and variance=$N_0/2$.

An M-ary communication system is shown next to illustrate the concept of DCMC channel model.



We consider the vector x_m to be a set of M-ary symbols (BPSK, M-PSK, M-QAM) that we wish to transmit. The symbols take on values from a set of M constellation signaling points defined by the M-ary scheme. The symbols generated are multiplied by orthogonal basis functions defined Φ (For two dimensional signal constellation, sine and cos are the usual basis functions). The multiplied signals are summed up to form a single symbol with duration T. Note that the symbols are limited in bandwidth (B) and exist for a finite symbol duration T. At the receiver, the received signal r(t) is split into different correlated signals when they get multiplied by the orthogonal basis functions. The outputs of the correlators are sampled at symbol time T and finally a decision is made on the symbol from the 'y' samples.

The entire system from the multiplier in the transmitter to the sampler in the receiver is considered as a channel. This is because; each block in the system can contribute to the uncertainty of the decision. The output of the system is viewed as a variable that can take any value from +∞ to -∞ (denoted by subscript j). The remaining system in between the input and output is considered as a variable with transition probability defined by $P(y|x_m)$.Then, the transition probability statistic is given by

$$P(\bar{y}|\bar{x}_m) = \prod_{n=1}^{N} \frac{1}{\sqrt{\pi N_0}} exp\left\{\frac{-(y_n - x_{mn})^2}{N_0}\right\}$$

$$= \frac{1}{\left(\sqrt{\pi N_0}\right)^N} exp\left\{\sum_{n=1}^{N} \frac{-(y_n - x_{mn})^2}{N_0}\right\} = \frac{1}{\left(\sqrt{\pi N_0}\right)^N} exp\left\{\frac{-|\bar{y} - \bar{x}_m|^2}{N_0}\right\}$$

Assuming that the input symbols occur with equal probability

$$P(\bar{x}_m) = \frac{1}{M} \quad ; m = 1,2, \dots, M$$

The generic channel capacity for DCMC channel is re-written in terms of transition probability as,

$$
\begin{aligned}
C &= \max_{P(\bar{x}_1),\dots,P(\bar{x}_M)} \sum_{m=1}^{M} \int_{N-Fold}^{-\infty} \dots \int_{-\infty}^{\infty} P(\bar{y}|\bar{x}_m) P(\bar{x}_m) \log_2 \left[\frac{P(\bar{y}|\bar{x}_m)}{\sum_{i=1}^{M} P(\bar{y}|\bar{x}_i) P(\bar{x}_i)} \right] d\bar{y} \\
&= \max_{P(\bar{x}_1),\dots,P(\bar{x}_M)} \sum_{m=1}^{M} \int_{N-Fold}^{-\infty} \dots \int_{-\infty}^{\infty} P(\bar{y}|\bar{x}_m) P(\bar{x}_m) \log_2 [P(\bar{y}|\bar{x}_m)] \, d\bar{y} \\
&\quad - \max_{P(\bar{x}_1),\dots,P(\bar{x}_M)} \sum_{m=1}^{M} \int_{N-Fold}^{-\infty} \dots \int_{-\infty}^{\infty} P(\bar{y}|\bar{x}_m) P(\bar{x}_m) \log_2 \left[\sum_{i=1}^{M} P(\bar{y}|\bar{x}_i) P(\bar{x}_i) \right] d\bar{y} \qquad (1) \\
&= A_1 - A_2
\end{aligned}
$$

Solving for A_1,

$$
\begin{aligned}
A_1 &= \sum_{m=1}^{M} \int_{N-Fold}^{-\infty} \dots \int_{-\infty}^{\infty} P(\bar{y}|\bar{x}_m) P(\bar{x}_m) \log_2 [P(\bar{y}|\bar{x}_m)] \, d\bar{y} \\
&= \frac{1}{M} \sum_{m=1}^{M} \int_{N-Fold}^{-\infty} \dots \int_{-\infty}^{\infty} \frac{1}{\left(\sqrt{\pi N_0}\right)^N} exp \left\{ \frac{-|\bar{y} - \bar{x}_m|^2}{N_0} \right\} \log_2 \left[\frac{1}{\left(\sqrt{\pi N_0}\right)^N} exp \left\{ \frac{-|\bar{y} - \bar{x}_m|^2}{N_0} \right\} \right] d\bar{y}
\end{aligned}
$$

Let

$$
\frac{y_n - x_{mn}}{\sqrt{N_0}} = k_n
$$

Then

$$
\frac{|\bar{y} - \bar{x}_m|^2}{N_0} = \frac{1}{N_0} \sum_{n=1}^{N} (y_n - x_{mn})^2 = \sum_{n=1}^{N} \left(\frac{y_n - x_{mn}}{\sqrt{N_0}} \right)^2 = \sum_{n=1}^{N} k_n^2 = |\bar{k}|^2
$$

and

$$
\frac{dy_n}{\sqrt{N_0}} = dk_n \;\Rightarrow\; d\bar{y} = \left(\sqrt{N_0}\right)^N d\bar{k}
$$

By the change of variables method, A_1 can be written as,

$$A_1 = \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\left(\sqrt{\frac{N_0}{\pi N_0}}\right)^N exp\left\{-|\bar{k}|^2\right\}\log_2\left[\frac{1}{\left(\sqrt{\pi N_0}\right)^N}exp\left\{-|\bar{k}|^2\right\}\right]d\bar{k}$$

$$= \int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\left(\frac{1}{\sqrt{\pi}}\right)^N exp\left\{-|\bar{k}|^2\right\}\log_2\left[\frac{1}{\left(\sqrt{\pi N_0}\right)^N}\right]d\bar{k}$$

$$- \int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\left(\frac{1}{\sqrt{\pi}}\right)^N exp\left\{-|\bar{k}|^2\right\}\log_2\left[exp\left\{-|\bar{k}|^2\right\}\right]d\bar{k}$$

$$= \log_2\left[\frac{1}{\left(\sqrt{\pi N_0}\right)^N}\right]\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\left(\frac{1}{\sqrt{\pi}}\right)^N exp\left\{-|\bar{k}|^2\right\}d\bar{k}$$

$$- \log_2 e\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\frac{|\bar{k}|^2}{\left(\sqrt{\pi}\right)^N}exp\left\{-|\bar{k}|^2\right\}d\bar{k}$$

$$= \log_2\left[\frac{1}{\left(\sqrt{\pi N_0}\right)^N}\right]\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\left(\frac{1}{\sqrt{\pi}}\right)^N\left[exp\left\{-k_1^2\right\}\ldots exp\left\{-k_N^2\right\}\right]d\bar{k}$$

$$- \log_2 e\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\frac{\left(k_1^2+\cdots+k_N^2\right)}{\left(\sqrt{\pi}\right)^N}exp\left\{-k_1^2\right\}\ldots exp\left\{-k_N^2\right\}d\bar{k}$$

$$= \log_2\left[\frac{1}{\left(\sqrt{\pi N_0}\right)^N}\right]\left[\int_{-\infty}^{\infty}\frac{1}{\sqrt{\pi}}exp(-k^2)dt\right]^N$$

$$- \log_2 e\left[\int_{-\infty}^{\infty}\frac{k_1^2}{\sqrt{\pi}}exp(-k_1^2)dk_1\right]\left[\int_{-\infty}^{\infty}\frac{1}{\sqrt{\pi}}exp(-k_2^2)dk_2\right]^{N-1}$$

Following identities are used to simplify the above equation,

$$\int_{-\infty}^{\infty}\frac{1}{\sqrt{\pi}}exp[\ \ k^2]dk = 1 \quad and \quad \int_{-\infty}^{\infty}\frac{1}{\sqrt{\pi}}k^2exp[\ \ k^2]dk = \frac{1}{2}$$

Therefore,

$$A_1 = -\frac{N}{2}\log_2(\pi N_0) - \frac{N}{2}\log_2 e = -\frac{N}{2}\log_2(e\pi N_0) \qquad (2)$$

Now solving for A_2

$$A_2 = \sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}P(\bar{y}|\bar{x}_m)P(\bar{x}_m)\log_2\left[\sum_{i=1}^{M}P(\bar{y}|\bar{x}_i)P(\bar{x}_i)\right]d\bar{y}$$

$$= \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}P(\bar{y}|\bar{x}_m)\log_2\left[\frac{1}{M}\sum_{i=1}^{M}P(\bar{y}|\bar{x}_i)\right]d\bar{y}$$

$$= \frac{1}{M}\log_2\left(\frac{1}{M}\right)\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}P(\bar{y}|\bar{x}_m)\,d\bar{y}$$

$$+ \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}P(\bar{y}|\bar{x}_m)\log_2\left[\sum_{i=1}^{M}P(\bar{y}|\bar{x}_i)\right]d\bar{y}$$

$$= \frac{1}{M}\log_2\left(\frac{1}{M}\right).M + \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}P(\bar{y}|\bar{x}_m)\log_2\left[\sum_{i=1}^{M}P(\bar{y}|\bar{x}_i)\right]d\bar{y}$$

$$= -\log_2(M) + A'_2 \qquad (3)$$

Let us solve for A'_2. For that

$$P(\bar{y}|\bar{x}_m) = \frac{1}{\left(\sqrt{\pi N_0}\right)^N} exp\left\{\frac{-|\bar{y}-\bar{x}_m|^2}{N_0}\right\}$$

Let

$$\frac{|\bar{y}-\bar{x}_m|^2}{N_0} = |\bar{k}|^2 \Rightarrow d\bar{y} = \left(\sqrt{N_0}\right)^N d\bar{k} \qquad (4)$$

Expanding the term by making substitutions,

$$\frac{y_n - x_{jn}}{\sqrt{N_0}} = \frac{y_n - x_{mn} + x_{mn} - x_{jn}}{\sqrt{N_0}} = \frac{y_n - x_{mn}}{\sqrt{N_0}} + \frac{x_{mn} - x_{jn}}{\sqrt{N_0}} = k_n + \frac{x_{mn} - x_{jn}}{\sqrt{N_0}}$$

$$\frac{(y_n - x_{jn})^2}{N_0} = {k_n}^2 + \frac{2k_n(x_{mn} - x_{jn})}{\sqrt{N_0}} + \frac{(x_{mn} - x_{jn})^2}{N_0} \qquad (5)$$

$$\begin{aligned}\frac{|\bar{y}-\bar{x}_j|^2}{N_0} &= \sum_{n=1}^{N} \frac{(y_n - x_{jn})^2}{N_0} \\ &= \sum_{n=1}^{N} {k_n}^2 + \sum_{n=1}^{N} \frac{2k_n(x_{mn} - x_{jn})}{\sqrt{N_0}} + \sum_{n=1}^{N} \frac{(x_{mn} - x_{jn})^2}{N_0} \\ &= |\bar{k}|^2 + \frac{2\bar{k}(\bar{x}_m - \bar{x}_j)}{\sqrt{N_0}} + \frac{(\bar{x}_m - \bar{x}_j)^2}{N_0} \qquad (6)\end{aligned}$$

Let's define the vector,

$$\bar{q}_{mj} = \left[\frac{(\bar{x}_{m1} - \bar{x}_{j1})}{\sqrt{N_0}}, \ldots, \frac{(\bar{x}_{mN} - \bar{x}_{jN})}{\sqrt{N_0}}\right]$$

Then the equation (6) can be written as,

$$\frac{|\bar{y}-\bar{x}_j|^2}{N_0} = |\bar{k}|^2 + 2\bar{k}\bar{q}_{mj} + |\bar{q}_{mj}|^2 \qquad (7)$$

Now A'_2 can be written as,

$$A'_2 = \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty} P(\bar{y}|\bar{x}_m)\log_2\left[\sum_{j=1}^{M} P(\bar{y}|\bar{x}_j)\right] d\bar{y}$$

$$= \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty} \frac{1}{(\sqrt{\pi N_0})^N} exp\left\{\frac{-|\bar{y}-\bar{x}_m|^2}{N_0}\right\}$$

$$\times \log_2\left[\sum_{j=1}^{M}\frac{1}{(\sqrt{\pi N_0})^N} exp\left\{\frac{-|\bar{y}-\bar{x}_m|^2}{N_0}\right\}\right] d\bar{y}$$

$$= \frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty} \frac{(\sqrt{N_0})^N}{(\sqrt{\pi N_0})^N} exp\left\{-|\bar{k}|^2\right\}$$

$$\times \log_2\left[\frac{1}{(\sqrt{\pi N_0})^N}\sum_{j=1}^{M} exp\left[-\left(|\bar{k}|^2 + 2\bar{k}\bar{q}_{mj} + |\bar{q}_{mj}|^2\right)\right]\right] d\bar{k}$$

$$= \frac{1}{M}\sum_{m=1}^{M}\log_2\left[\frac{1}{(\sqrt{\pi N_0})^N}\right]\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\frac{1}{(\sqrt{\pi})^N} exp\left\{-|\bar{k}|^2\right\} d\bar{k}$$

$$+\frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\frac{1}{(\sqrt{\pi})^N} exp\left\{-|\bar{k}|^2\right\}\log_2\left[exp\left\{-|\bar{k}|^2\right\}\right] d\bar{k}$$

$$+\frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\frac{1}{(\sqrt{\pi})^N} exp\left\{-|\bar{k}|^2\right\}\log_2\left[\sum_{j=1}^{M} exp\left\{-2\bar{k}\bar{q}_{mj} - |\bar{q}_{mj}|^2\right\}\right] d\bar{k}$$

Using the similar method to derive equation (2), A'_2 can be written as,

$$A'_2 = -\frac{N}{2}\log_2(e\pi N_0)$$

$$+\frac{1}{M}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty}\frac{1}{(\sqrt{\pi})^N} exp\left\{-|\bar{k}|^2\right\}\log_2\left[\sum_{j=1}^{M} exp\left\{-2\bar{k}\bar{q}_{mj} - |\bar{q}_{mj}|^2\right\}\right] d\bar{k} \quad (8)$$

From (1),(2),(3) and (8), the capacity of the DCMC model is given by

$$C = \log_2(M) \quad (9)$$

$$-\frac{1}{M(\sqrt{\pi})^N}\sum_{m=1}^{M}\int_{N-Fold}^{-\infty}\cdots\int_{-\infty}^{\infty} exp\left\{-|\bar{k}|^2\right\}\log_2\left[\sum_{j=1}^{M} exp\left\{-2\bar{k}\bar{q}_{mj} - |\bar{q}_{mj}|^2\right\}\right] d\bar{k}$$

Where the vector,

$$\bar{q}_{mj} = \left[\frac{(\bar{x}_{m1}-\bar{x}_{j1})}{\sqrt{N_0}},\ldots,\frac{(\bar{x}_{mN}-\bar{x}_{jN})}{\sqrt{N_0}}\right] = \frac{(\bar{x}_m - \bar{x}_j)}{\sqrt{N_0}}$$

*****

Back to Table of Contents

A4: Natural and Binary Codes

In a given communication system, we always want to send data that represent some real world physical quantity (example: speech, temperature, etc..,) .The real world physical quantity exists in analog domain. If we want to send it via a digital communication system, it has to be converted to digital domain. The above process of converting from analog to digital domain is done by an Analog to Digital Converter (ADC). A typical ADC contains the following blocks: 1) Sampler 2) Quantizer and 3) Encoder.

The Sampler converts the analog signal (manifestation of the real world physical quantity) to a sequence of discrete samples that are uniformly spaced in time. For more details on sampling process, refer sections 1.3-1.4 – (Sampling Theorem – Baseband Sampling and Sampling Theorem – bandpass or under sampling)

The sampled signals from the sampling process are discrete in time but their amplitudes are still continuous. In order to have a full representation of a signal in digital form, the signal has to be discrete in both time and amplitude. The amplitude of the sampled signal can take on values from an infinite number of possible amplitude levels. These infinite numbers of possible amplitudes have to be mapped to manageable amplitude levels by mapping different ranges of amplitude levels to a finite set of discrete amplitude levels. This requires the next block in the ADC - namely the quantizer.

The quantizer discretizes the continuous amplitude of the sampled signal to a finite set of discrete levels. Several types of quantizers – uniform, non-uniform (A-law quantizer, μ-law quantizer [TIAlawMuLaw]), differential quantizer, etc. exist to fulfill the purpose.

We are not done yet. Thus-far, the signal is just discretized in amplitude and time. The quantized signal has to be represented in some numeric form to get a digital representation. The encoder, the next block in the ADC, maps the discrete amplitude levels of the quantized samples to codewords. Codewords are just some form of numeric representation that the encoder assigns to each discretized amplitude level. Two of the convenient methods for one-to-one mapping of amplitude levels to codewords include: 1) Natural Binary Coding and 2) Gray coding. In natural binary mapping the discrete amplitude levels are coded in binary format.

Natural Binary Coding:

Consider that the sampler has discretized the samples in time and the quantizer maps discretized ranges of voltages to discrete amplitude levels. The next task is to convert the discrete range of amplitude levels to digital representation using an encoder. The first column in the following tables gives the encoder output in binary. The second column gives possible interpretations. Four commonly known interpretations exists namely – unsigned, signed magnitude, 1's complement and 2's complement form. The encoder just spits out the binary pattern and it is up to the designer to decide how to interpret the binary pattern in order to represent the quantized values. The interpretation influences the quantizer ranges that will be mapped to specific values. Quantizer design and encoder's output interpretation always go hand in hand.

Out of the four common interpretations listed above, unsigned integer representation can represent positive values only. The other three representations can accommodate both positive and negative values.

In the signed magnitude representation, also called Folded Binary Code / Foldover Binary Code, the Most Significant Bit (MSB) of the binary pattern represents the sign of the number (positive/negative)

and the rest of the bits represent the magnitude. In 1's complement representation, the negative values are just the 1's complement of positive values. To convert a binary representation to 1's complement, one just has to flip all the bits (1's to 0's and 0's to 1's). To convert to 2's complement form, convert the binary pattern to 1's complement and then add '1' to the resulting output.

The designer can conveniently choose any of the above mentioned forms to interpret the binary output of the encoder. For example, if the designer interprets it as 1's complement or signed magnitude representation, then he would end-up in having two different representations for the value '0'. All the further calculations in DSP has to be done by keeping this fact in mind. Having two representations for '0' poses greater threat to the reliability of design. To avoid this ambiguity, 2's complement representation is always the choice of interpretation. Additionally two's complement interpretation results in faster and simpler hardware. Also it can be noted that, from the 2's complement example given below, given the same number of bits (3 bits) to represent the binary output of the encoder, the two's complement encoding can represent voltage ranges from -4.5V to +3.5V without any ambiguity. But in the case of signed magnitude and one's complement encoding the range of representable voltage ranges shrinks to -3.5V to +3.5V but they also result in ambiguity in representing 0V.

Bit pattern (Binary representation)	Interpretation (Signed Magnitude/ Folded Binary Code)	Quantized Value (Volts)	Sample Quantization Range - (Volts)
011	+3	+3	+2.5V to +3.5V
010	+2	+2	+1.5V to +2.5V
001	+1	+1	+0.5V to +1.5V
000	+0	+0	+0V to +0.5V
111	-3	-3	-3.5V to -2.5V
110	-2	-2	-2.5V to -1.5V
101	-1	-1	-1.5V to -0.5V
100	-0	-0	-0.5V to -0V

Bit pattern (Binary representation)	Interpretation (1's complement)	Quantized Value (Volts)	Sample Quantization Range - (Volts)
011	+3	+3	+2.5V to +3.5V
010	+2	+2	+1.5V to +2.5V
001	+1	+1	+0.5V to +1.5V
000	0	+0	+0V to +0.5V
111	0	-0	-0V to -0.5V
110	-1	-1	-0.5V to -1.5V
101	-2	-2	-1.5V to -2.5V
100	-3	-3	-2.5V to -3.5V

Bit pattern (Binary representation)	Interpretation (2's complement)	Quantized Value (Volts)	Sample Quantization Range - (Volts)
011	+3	+3	+2.5V to +3.5V
010	+2	+2	+1.5V to +2.5V
001	+1	+1	+0.5V to +1.5V
000	0	0	-0.5V to +0.5V
111	-1	-1	-1.5V to -0.5V
110	-2	-2	-2.5V to -1.5V
101	-3	-3	-3.5V to -2.5V
100	-4	-4	-4.5V to -3.5V

Gray Coding:

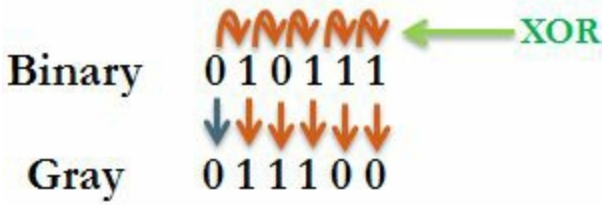
Gray Coding is another form of representation that is used ubiquitously in all applications. In Gray coding, the adjacent representations (symbols) differ by only one bit. Gray coding, when combined with Forward Error Correction codes capable of corrective single bit errors, can aid in the correction of erroneous reception of bits that spill onto adjacent symbols. Digital modulation techniques like M-PSK and M-QAM use Gray coding representation to represent the symbols that are modulated.

Converting from Natural Binary to Gray:

To convert a binary representation ($X_3\ X_2\ X_1\ X_0$) to Gray code ($Y_3\ Y_2\ Y_1\ Y_0$), following method can be used.

$$
\begin{aligned}
Y_3 &= X_3 \\
Y_2 &= X_3 \oplus X_2 \\
Y_1 &= X_2 \oplus X_1 \\
Y_0 &= X_1 \oplus X_0
\end{aligned}
$$

That is, the MSB (Y_3) is same for both binary and Gray codes. The next bit (Y_2) is the XOR of previous bit (X_3) and the present bit (X_2) of the binary code and so on. Following example illustrates this concept using a 6-bit code.



Converting from Gray to Natural Binary:

To convert a Gray code representation ($Y_3\ Y_2\ Y_1\ Y_0$) to binary code ($X_3\ X_2\ X_1\ X_0$), following method can be used.

$$
\begin{aligned}
X_3 &= Y_3 \\
X_2 &= Y_2 \oplus X_3 \\
X_1 &= Y_1 \oplus X_2 \\
X_0 &= Y_0 \oplus X_1
\end{aligned}
$$

That is, the MSB (X_3) of binary code is same as that of gray code. The next bit (X_2) is the XOR of previous result (X_3) and the present bit (Y_2) of the gray code and so on. Following example illustrates this concept using 6-bit code.

The following table illustrates the conversion for a three bit system.

Decimal Values	Natural Binary Code	Gray Code	Gray Code values
0	000	000	0
1	001	001	1
2	010	011	3
3	011	010	2
4	100	110	6
5	101	111	7
6	110	101	5
7	111	100	4

Matlab Code:

Matlab Code to convert decimal values directly to Gray

```
function [grayCoded]=dec2gray(decimalInput)

[rows,cols]=size(decimalInput);
grayCoded=zeros(rows,cols);
for i=1:rows
    for j=1:cols
        grayCoded(i,j)=bitxor(bitshift(decimalInput(i,j),-1),decimalInput(i,j));
    end
end
end
```

Matlab Code to convert Gray to decimal values

```
function [decimal]=gray2dec(grayInput)

grayInput=uint8(grayInput); %Force datatype to uint8
[rows,cols]=size(grayInput);
decimal=zeros(rows,cols);

for i=1:rows
    for j=1:cols
        temp = bitxor(grayInput(i,j),bitshift(grayInput(i,j),-8));
        temp = bitxor(temp,bitshift(temp,-4));
```

```
        temp = bitxor(temp,bitshift(temp,-2));
        temp = bitxor(temp,bitshift(temp,-1));
        decimal(i,j) = temp;
    end
end
end
```

Sample Run:

```
>> decimalInput=[0 1 2 3 4 5 6 7]
decimalInput = 0 1 2 3 4 5 6 7

>> graycoded=dec2gray(decimalInput)
graycoded = 0 1 3 2 6 7 5 4

>> graycoded=gray2dec(graycoded)
graycoded = 0 1 2 3 4 5 6 7
```

*****

Back to Table of Contents

A5: Constructing a rectangular constellation for 16QAM

Any rectangular QAM constellation is equivalent to superimposing two Amplitude Shift Keying (ASK) signals on quadrature carriers (I and Q components). For a 4-QAM modulation, each symbol is of size $k = \log_2(M) = \log_2(4) = 2$ bits. For a 16-QAM modulation, the symbol size is $k = \log_2(16) = 4$ bits.

There exist other constellation shapes (like circular, triangular constellations) that are more efficient (in terms of energy required to achieve same the error probability) than the standard rectangular constellation. Rectangular constellations are the preferred choice of implementation due to its simplicity in implementing modulation and demodulation.

In any M-QAM constellation, in order to restrict the erroneous symbol decisions to single bit errors, the adjacent symbols in the transmitter constellation should not differ by more than one bit. This is usually achieved by converting the input symbols to Gray coded symbols (refer Appendix A4) and then mapping it to the desired QAM constellation. But this intermediate step can be skipped altogether by using a Look-Up-Table (LUT) approach which properly translates the input symbol to appropriate position in the constellation. We will exploit the inherent property of Karnaugh Maps to generate the look-up table for the gray coded 16-QAM constellation

Karnaugh Maps:

If you are familiar with Karnaugh Maps (K-Maps) used in digital electronics, it is easier for you to identify that the K-Maps are constructed based on Gray Codes.

To generate a gray coded 16-QAM constellation we will use a 4 variable (since for 16 QAM the

symbol size is 4 bits) K-Map. A 16 QAM constellation can be constructed using two 4-ASK signals. For a 4-ASK signal there are 4 amplitude levels. In our case the 4 amplitude levels are {-3,-1, +1, +3}

A 4 variable K-Map will look like the one shown below. Here ABCD are the 4 variables appearing in the bit order (MSB at A and LSB at D).

AB\CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Now replace the variables AB and CD with the 4 amplitude levels {-3,-1, +1, +3}. Here AB consists of the two bits that are ASK modulated on the in-phase arm and CD consists of the two bits that are ASK modulated on the quadrature arm.

AB\CD	-3	-1	+1	+3
-3	0000	0001	0011	0010
-1	0100	0101	0111	0110
+1	1100	1101	1111	1110
+3	1000	1001	1011	1010

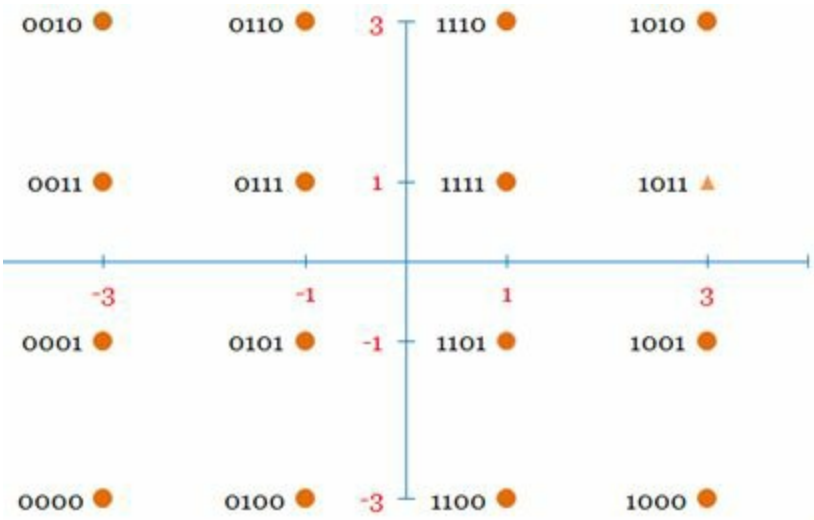
Now assign equivalent decimal values (shown in square brackets) to each cell value in the table.

AB\CD	-3	-1	+1	+3
-3	0000 [0]	0001 [1]	0011 [3]	0010 [2]
-1	0100 [4]	0101 [5]	0111 [7]	0110 [6]
+1	1100 [12]	1101 [13]	1111 [15]	1110 [14]
+3	1000 [8]	1001 [9]	1011 [11]	1010 [10]

Construct an array with the numbers indicated in the square brackets as array indices. The respective value for each index in the array is taken as AB + j CD. The array indices are the input to the 16-QAM modulator and the corresponding array values are the 16-QAM modulator output that are gray coded and mapped to constellation points.

Array Index (Input Symbols to 16-QAM modulator)	Array Index Represented in Binary	Array Value -> AB + j CD (Output of 16-QAM modulator)
0	0000	-3 - 3j
1	0001	-3 - 1j
2	0010	-3 + 3j
3	0011	-3 + 1j
4	0100	-1 -3j
5	0101	-1 -1j
6	0110	-1 +3j
7	0111	-1 +1j
8	1000	+3 -3j
9	1001	+3 -1j
10	1010	+3 +3j
11	1011	+3 +1j
12	1100	+1 - 3j
13	1101	+1 - 1j
14	1110	+1 +3j
15	1111	+1 +1j

Let us draw the constellation to verify the approach. Plot the in-phase and quadrature phase components (QAM output), and each point with the equivalent binary representation of the array index gives the 16-QAM constellation where the adjacent symbols differ by only one bit.



The same approach can be used to construct rectangular constellation for QAM modulation for any values of M. For example a 64-QAM constellation would require a $k=\log_2(64) = 6$ Variable Karnaugh Map to begin with.

*****

Back to Table of Contents

A6: Q Function and Error Function

Q functions are often encountered in the theoretical equations for calculating Bit Error Rate (BER) involving AWGN channel. A brief discussion on Q function and its relation to "erfc" function is given here.

The probability that a continuous random variable X takes a value less than or equal to some quantity x_0 (i.e, $P(X \leq x_0)$ is given by Continuous Density Function (CDF)

Then we can define another function called Complementary Continuous Density Function (CCDF) which is the complementary of the CDF. It gives the probability that $P(X > x_0)$. CDF and CCDF are complementary because they add up to unity.

$$P(X \leq x_0) + P(X > x_0) = 1$$

$$P(X > x_0) = 1 - P(X \leq x_0)$$

The quantity $P(X > x_0)$ is called upper tail probability. This is because; it gives the probability of the tail region on the right side (upper part) of the PDF curve (See the next figure).

Thus, we can see that the upper tail probability is equal to one minus CDF. In general, tail probability is the probability that a random variable deviates by a given amount from its expectation.

Let us derive the CDF of a standard normal distribution and extend the concept to finding its tail probability.

For a standard normal distribution $X \sim N(\mu = 0, \sigma^2 = 1)$, the PDF is given by

$$f_X(x) = \frac{1}{\sqrt{2\pi}} exp\left[-\frac{x^2}{2}\right]$$

The CDF can be calculated from PDF as,

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(x)dx = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-x^2/2} dx$$

Then the CCDF (complementary CDF) is given by

$$\bar{F}_X(x) = 1 - F_X(x) = P(X > x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{x^2}{2}} dx$$

The Complementary CDF of the standard normal distribution, defined above, gives the tail probability. This is otherwise called Q function.

$$Q(x) = P(X > x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{x^2}{2}} dx \qquad -(1)$$

Thus, Q function is used to calculate the tail probability of standard normal distribution X~N(0,1).

What if we have a non-standard normal distribution and we want to calculate its tail probability? We need to transform the non-standard distribution to standard distribution and calculate the tail probability.

A standard normal random variable $X_s \sim N(0,1)$ can be converted to non-standard normal variable with

arbitrary mean and sigma - $X_{NS} \sim N(\mu, \sigma^2)$ as

$$X_{NS} = \sigma . X_S + \mu$$

Take the standard normal random variable, multiply it by the given sigma (standard deviation) and add the given mean. In this way, a normal random variable with arbitrary mean and standard deviation can be generated from standard normal random variable.

Applying the transformation in the reverse manner, one can translate the non-standard random variable to standard domain as

$$X_S = \frac{X_{NS} - \mu}{\sigma}$$

Let us apply this transformation to find the tail probability of a normal random variable $X \sim N(\mu, \sigma^2)$. This transformation is often written in terms of another variable as

$$Z = \frac{X - \mu}{\sigma} \qquad -(2)$$

Here Z is the standard version of the normal distribution.

The tail probability is given by

$$P(X > x) = P\left(\frac{X - \mu}{\sigma} > \frac{x - \mu}{\sigma}\right) = P\left(Z > \frac{x - \mu}{\sigma}\right) \qquad -(3)$$

Then from equations (1), (2) and (3), the Q function is given by

$$Q(z) = Q\left(\frac{x - \mu}{\sigma}\right) = P\left(Z > \frac{x - \mu}{\sigma}\right) = \frac{1}{\sqrt{2\pi}} \int_{(x-\mu)/\sigma}^{\infty} e^{-\frac{z^2}{2}} dz \quad -(4)$$

Some important results:

Keep a note of the following equations that can come handy when deriving probability of bit errors for various scenarios. These equations are compiled here for easy reference.

If we have a random variable $X \sim N(\mu, \sigma^2)$, the probability that X>x is given by

$$P(X > x) = Q\left(\frac{x - \mu}{\sigma}\right)$$

If we want to know the probability of X to be away from its mean μ by at least 'a' (either to the left or to the right of the PDF curve), then

$$P(X > \mu + a) = P(X < \mu - a) = Q\left(\frac{a}{\sigma}\right)$$

The probability of X to be away from its mean μ by at least 'a' (if it does not matter in which side of the curve), is given by

$$P(X > \mu + a) \text{ or } P(X < \mu - a) = 2.Q\left(\frac{a}{\sigma}\right)$$

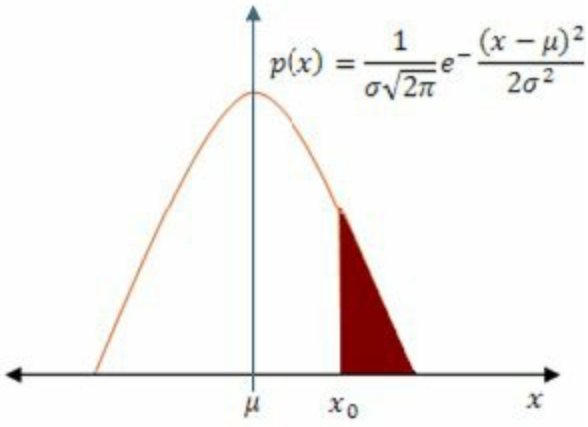
Application of Q function:

Gaussian process is the underlying model for an AWGN channel. The probability density function of

Gaussian distribution is given by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Generally, in BER derivations, the probability that a Gaussian Random Variable $X \sim N(\mu, \sigma^2)$ exceeds some value X_0 - is evaluated as the area of the shaded region as shown in the following figure. Q function is utilized to calculate such tail probabilities. To know how Q function is applied in computing the Bit Error Rate (BER) of a BPSK modulation system, refer section 6.12.



Error function and Complementary Error Function:

The error function represents the probability that the parameter of interest is within a range between $-x/o\sqrt{2}$ and $+x/o\sqrt{2}$. The "complementary" error function gives the probability that the parameter lies outside that range.

The error function is given by

$$\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-x^2} dx \qquad (5)$$

and the complementary error function is given by

$$\text{erfc}(z) = 1 - \text{erf}(z) \qquad (6)$$

or equivalently,

$$\text{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-x^2} dx \qquad (7)$$

Q function and Complementary Error Function (erfc):

From the limits of the integrals in equation (1) and (7) one can conclude that Q function is directly related to complementary error function (erfc).

It follows from equation (1) and (7), Q function is related to complementary error function by the following relation.

$$Q(z) = \frac{1}{2} erfc\left(\frac{z}{\sqrt{2}}\right) \qquad (8)$$

*****

Back to Table of Contents

References

[3GPPTS25104] 3GPP Specification - 3GPP TS 25.104 V6.8.0 (2004-12) - http://www.3gpp.org/specifications

[AgilentNearFar] Theory behind Near-field and Far-field - http://www.home.agilent.com/upload/cmc_upload/All/NSI-near-far.pdf?&cc=SG&lc=eng

[Amstrong1938] E. H. Armstrong: “A Method of Reducing Disturbances in Radio Signaling by a System of Frequency-Modulation,” Proc. IRE, 24, pp. 689-740, May, 1936

[Blahut1987] Blahut R. E., Principles and Practice of Information Theory, MA,Addison–Wesley, 1987.

[Campanella1990] Campanella et.al, “On the Channel Capacity for Constant Envelope Signals with Effective Bandwidth Constraint,” IEEE Transactions on Communication, vol. 38, pp.1164–1172, Aug. 1990.

[Deng2003] Deng, Li & Douglas O'Shaughnessy, "Speech processing: a dynamic and optimization-oriented approach", CRC Press, First edition, 2003 pp. 41–48. ISBN 0-8247-4040-8

[Engelberg2004] Engelberg, “The central limit theorem and low-pass filters”, Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 13-15 Dec. 2004, pp-65-68.

[Fano1949] Fano, R.M., “The transmission of information”, Technical Report No. 65 (Cambridge (Mass.), USA, Research Laboratory of Electronics at MIT).

[Gallager1968] R. G. Gallager, “Information Theory and Reliable Communication”, NewYork, Wiley, 1968.

[Garrabrant1989] Garrabrant et.al, “Trellis Coded Modulation Applied to Noncoherent Detection of Orthogonal Signals,” MILCOM ’89, vol. 3, pp. 774–778, Oct. 1989.

[GoldCode] Gold. R., “Optimal binary sequences for spread spectrum multiplexing”, IEEE Transactions on Information Theory, 13 (4), pp. 619–621

[Goldsmith2005] Andrea Goldsmith, "Wireless Communications," Cambridge University Press, 2005,644 pp.

[Golomb1966] S. W. Golomb,"Run-length encoding",IEEE Trans Info Theory 12(3),July 1966,pp. 399

[HCChen] HC Chen et al., “Moving Average Filter with its application to QRS detection”, IEEE Computers in Cardiology, 2003 - http://www.cinc.org/archives/2003/pdf/585.pdf

[Heller1971] Heller, J. Jacobs, I., “Viterbi Decoding for Satellite and Space Communication”, IEEE Transactions on Communication Technology, Volume: 19, Issue: 5, Part 1,pp: 835-848, October 1971

[Huff1952] Huffman D.A, Proceedings of the I.R.E., September 1952, pp. 1098-1102.

[IEEE80211] IEEE 802.11 specification – “Orthogonal frequency division multiplexing (OFDM) PHY specification for the 5 GHz band” – chapter 17 - http://standards.ieee.org/getieee802/download/802.11-2007.pdf

[Lindsey1964] Lindsey, W. C., "Error probabilities for Rician fading multichannel reception of binary and N-ary signals", IEEE Trans. Inform. Theory, Vol. IT-10, pp. 339-350,Oct 1964

[MatlabRand] Matlab Documentation entry on generating uniformly distributed pseudorandom numbers using rand function - http://www.mathworks.in/help/techdoc/ref/rand.html

[MatlabRandn] Matlab Documentation entry on generating Gaussian distributed pseudorandom numbers using randn function - http://www.mathworks.in/help/techdoc/ref/randn.html

[MITOCWFilter] Filtering – OpenCourseWare – MIT - http://ocw.mit.edu/resources/res-6-007-signals-and-systems-spring-2011/video-lectures/lecture-12-filtering/

[NatNapoletano] Bayes' theorem for everyone - Youtube video - by Nat Napoletano - http://www.youtube.com/watch?v=XR1zovKxilw

[Pasupathy1989] Pasupathy.S "Glories of Gaussianity”, (Light Traffic)", IEEE Communications magazine, August 1989, p.38 - http://www.comm.toronto.edu/~pas/articles/ltraffic/18_gloriesofgaussianity.pdf

[Proakis1969] J. G. Proakis, "Digital Communications Second Edition", New York, McGraw–Hill, 1989

[QualcommAN1652] Qualcomm Application Note AN1650-2, “Setting Soft-Decision Thresholds for Viterbi Decoder Code Words from PSK Modems”

[Rice1979] Rice R.F, "Some Practical Universal Noiseless Coding Techniques, "Jet Propulsion Laboratory, Pasadena, California, JPL Publication 79—22, Mar. 1979.

[Robertson1969] Robertson G.H, "Computation of the Noncentral Chi-Square Distribution", The Bell system technical journal, pp.202 January 1969 - http://www.ascend.com/bstj/vol48-1969/articles/bstj48-1-201.pdf

[Shannon1948] C. E. Shannon, “A Mathematical Theory of Communication”, Bell Syst. Techn. J., Vol. 27, pp.379-423, 623-656, July, October, 1948

[Shannon1949]Shannon, C.E., “ Communications in the Presence of Noise” Proc. IRE, Volume 37 no1, January 1949, pp10-21

[Sklar2001] Sklar, Bernard, Digital Communications, Prentice Hall, 2001.pp 525-529

[Smith1975] J. I. Smith, “A computer generated multipath fading simulation for mobile radio,” IEEE Trans. Veh. Technol., vol. VT-24, no. 3, pp. 39–40, Aug. 1975

[Srinivasa2009] Srinivasan, S.; Haenggi, M. "Path loss exponent estimation in large wireless networks", Information Theory and Applications Workshop, On pages 124 - 129, Feb 2009.

[TechnicalJar] How the addition of cyclic prefix converts linear convolution to circular convolution - http://www.technicaljar.com/?p=237

[TIAlawMuLaw] A-Law and mu-Law Companding Implementations Using the TMS320C54,TI Application Note: SPRA163A - http://www.ti.com/lit/an/spra163a/spra163a.pdf

[Trefethen1997] Lloyd N. Trefethen, David Bau III (1997), Numerical linear algebra, Philadelphia: Society for Industrial and Applied Mathematics, ISBN 978-0-89871-361-9, pp.56

[Ungerboeck1982] Ungerboeck G., “Channel Coding with Multilevel/Phase Signals,” IEEE Transactions Information Theory, vol. IT–28, pp. 56–67, Jan. 1982

[WebAndreas] Lecture by Andreas Gerstlauer-Convolutional (Viterbi) Encoding- http://users.ece.utexas.edu/~gerstl/ee382v-ics_f09/lectures/Viterbi.pdf

[WebComplex2Real] Inter Symbol Interference and Root Raised Cosine Filtering – Complex2real - http://www.complextoreal.com/chapters/isi.pdf

[WebITUG711] ITU-T Recommendation G.711 - http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.711-198811-I!!PDF-E&type=items

[WebKen] The care and feeding of digital, pulse-shaping filters – By Ken Gentile - http://www.feng.pucrs.br/~decastro/pdf/ShapingFilterDesign.pdf

[WebMichael] Michael Joost,”Theory of Root-Raised Cosine Filter”, Krefeld DE, December 2010 - http://www.michael-joost.de/rrcfilter.pdf

[WebMPEG] MPEG Group website http://mpeg.chiariglione.org/

[WebPurcell] Maximum Likelihood Estimation – a detailed explanation by S.Purcell - http://statgen.iop.kcl.ac.uk/bgim/mle/sslike_6.html

[WebRobert] Robert G. Maunder- 'Matlab DCMC capacity' - Rob Maunder's research pages - http://users.ecs.soton.ac.uk/rm/resources/matlabcapacity/

[WebScotts] The Scotts Guide to Electronics, “Information and Measurement”, University of Andrews - School of Physics and Astronomy - http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/iandm/part8/page1.html

[WebVladimir] Vladimir Stojanovic - Viterbi Algorithm Implementation - MIT courseware - http://www.core.org.cn/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-973Spring-2006/9A86BA46-AF46-4EE3-AF3B-797B61DA202A/0/lecture_14.pdf

[Welch1985] Welch; Terry A., “High speed data compression and decompression apparatus and method”, U.S. Patent 4,558,302, filed June 20 1983 and issued December 10 1985

[WikiColorNoise] Wiki Entry on Colors of Noise - http://en.wikipedia.org/wiki/Colors_of_noise

[William1993] William B. Pennebaker & Joan L. Mitchell,"JPEG still image data compression standard", Eighth Edition,Kluwer Academic Publishers,1993

[Wozencraft1965] Wozencraft et.al , Principles of Communications Engineering,. NewYork, Wiley, 1965.

[Young2000]D.J. Young and N.C. Beaulieu, "The generation of correlated Rayleigh random variates by inverse discrete Fourier transform," IEEE transactions on Communications, vol. 48, pp. 1114-1127, July 2000.

[ZivMay1977] Ziv, "IEEE Transactions on Information Theory", IT-23-3, May 1977, pp. 337-343.

[ZivSep1977] Ziv, "IEEE Transactions on Information Theory", IT-24-5, Sep. 1977, pp. 530-537.

Back to Table of Contents

##END##

About the Author

I am Mathuranathan Viswanthan, founder and author of GaussianWaves, a technical blog on signal processing and digital communication systems that enjoys a huge readership worldwide. I hold a Masters degree in Communication Engineering from India's premier technology university Birla Institute of Technology and Science - Pilani (fondly called as BITS-Pilani). I also have seven years of industry experience in the same field.

Connect with Me Online

Website: http://www.gaussianwaves.com
Facebook: http://www.facebook.com/gaussianwaves
Google+: http://plus.google.com/116172790949784103297/posts
Twitter: http://www.twitter.com/gaussianwaves
Smashwords: http://www.smashwords.com/profile/view/mathuranathan
Email: support@gaussianwaves.com